Sistema de búsqueda de respuestas adaptable a distintos dominios

Trabajo de fin de grado

Aitor Cayón Ruano
José Javier Cortés Tejada
Fernando Pérez Gutiérrez
Gabriel Sellés Salvà

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Curso académico 2017/2018

Memoria presentada para el trabajo de fin de grado de
Ingeniería Informática

Dirigida por el Doctor
Alberto Díaz
y codirigida por
Antonio F. García Sevilla

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Curso académico 2017/2018

Agradecimientos

Queremos agradecer la ayuda y el apoyo recibido por parte de un gran número de personas sin las cuales este proyecto no se habría llevado a cabo.

En primer lugar, queremos agradecer a Alberto Díaz Esteban y Antonio F. García Sevilla por ofrecernos este proyecto, por la confianza depositada en nuestras capacidades y por su colaboración y apoyo en todo momento.

También queremos agradecer especialmente el apoyo e interés mostrado por nuestros familiares y amigos, que han estado ahí cuando los hemos necesitado.

Por último, agradecer a los profesores, que nos han ayudado a alcanzar nuestras metas formándonos y proporcionándonos conocimiento día a día.

Resumen

En la actualidad, los sistemas de búsqueda de respuestas tienen un papel destacable tanto en la sociedad como en la industria. Debido a su utilidad, a lo largo de la historia reciente, se han realizado distintos tipos de implementaciones, cada una centrándose en su finalidad concreta. Aún así, la mayoría de implementaciones basan su funcionamiento en estudios estadísticos y modelos entrenados con gran volumen de datos.

En este proyecto hemos optado por implementar un sistema con un enfoque distinto: un sistema de búsqueda de respuestas que resuelva consultas mediante el análisis lingüístico de conocimiento textual.

Además, el sistema implementado, es adaptable a múltiples dominios. Esto implica que tiene la capacidad de utilizar distintas fuentes de información textual a la hora de responder a una consulta introducida por el usuario. Esta funcionalidad permite la resolución de consultas de distintos ámbitos y dominios, a la vez que facilita considerablemente la inserción de nuevas fuentes de información textual.

Para realizar esta implementación, este proyecto consta de varias partes diferenciadas:

- **Análisis de la consulta introducida:** se realiza un análisis sintáctico y semántico de la consulta introducida por el usuario.
- **Obtención información textual:** se buscan documentos que contengan las palabras clave de la consulta introducida y, por lo tanto, potencialmente contengan la respuesta a la consulta.
- **Obtención de la respuesta:** se busca en dichos documentos el fragmento que responde a la consulta.
- **Generación de la respuesta final:** se genera una sentencia en lenguaje natural que contiene la respuesta.

Por último, se ha ejecutado el sistema sobre un repositorio propio, BioASQ y Simple Wikipedia para evaluar las capacidades de este.

Abstract

At present, question answering systems have great relevance both in society and industry. Due to its utility, in recent history, they have been implemented in several ways, each of them with a specific purpose. Nevertheless, most of implementations are based on statistical studies or models trained with a large amount of data.

We have decided to implement our system with a different approach: a question answering system capable of solving queries through linguistic analysis using textual knowledge.

In addition, the implemented system can be adapted to multiple domains. This implies we can use different sources of textual information when answering the query introduced by the user. This functionality allows the resolution of queries from different fields and domains while considerably simplifying the insertion of new sources of textual information.

The implementation of this project has been divided in different parts:

- **Analysis of the introduced query:** executes the syntactic and semantic analysis of the query introduced by the user.
- **Textual information retrieval:** searches for the documents containing keywords from the introduced query and posibly containing the answer to it.
- **Answer retrieval:** searches for the fragment in the documents answering the query.
- **Generation of the final answer:** generates a natural language sentence containing the answer.

Finally, the system has been executed over our own repository, BioASQ and Simple Wikipedia in order to validate its capabilites.

Palabras clave

- Conocimiento textual
- Grafeno
- Spacy
- Neo4J
- Adaptable
- Múltiples dominios
- Búsqueda de respuestas
- Recuperación de información
- Procesamiento de lenguaje natural

Keywords

- Textual knowledge
- Grafeno
- Spacy
- Neo4J
- Adaptable
- Multiple domains
- Question Answering
- Information Retrieval
- Natural language processing

Índice

Índice de figuras

Capítulo 1

Introducción

Resumen: en este capítulo se expone la motivación que nos ha llevado a desarrollar este proyecto, así como los objetivos iniciales del mismo.

1.1. Motivación

A lo largo de la historia han existido varios sistemas de búsquedas de respuestas que han tratado de satisfacer las necesidades del ser humano, como, por ejemplo, los sistemas *online* de atención al cliente usados por las empresas de transporte público. Muchos de estos sistemas basan su funcionamiento en estudios estadísticos y modelos entrenados con gran volumen de datos para resolver estas cuestiones.

No siempre se dispone de tal cantidad de datos con los que entrenar un modelo por lo que han ido surgiendo distintas aproximaciones para resolver dicho problema (8). Una de las más interesantes es la basada en el análisis lingüístico de conocimiento textual.

Este enfoque permite la resolución del problema basándonos únicamente en la información contenida en diversos textos, independientemente del dominio al que pertenezcan tanto dichos textos como las consultas realizadas.

Los sistemas de búsqueda de respuestas no solo se distinguen por la forma de resolver el problema sino también según el dominio sobre el que trabajan (11). Varios sistemas están implementados para resolver las consultas pertenecientes a un dominio concreto (un sitio web, un repositorio específico, etc). Sin embargo, existen sistemas de múltiples dominios que aprovechando la gran cantidad de información existente en Internet son capaces de responder a preguntas de distintos ámbitos. Estos son especialmente interesantes

porque el abanico de problemas que pueden resolver es mucho mayor.

1.2. Objetivos

El objetivo principal de este TFG es la construcción de un sistema de búsqueda de respuestas que mediante el análisis lingüístico de conocimiento textual sea capaz de responder a preguntas introducidas por el usuario.

Para aprovechar la gran cantidad y diversidad de conocimiento textual que podemos encontrar en Internet, el sistema debe ser adaptable a diversos dominios. Dicho sistema debe construirse partiendo de una arquitectura genérica (4) que haga uso de distintas fuentes de conocimiento. Otro requisito que deberá cumplir el sistema es que sea fácilmente integrable con nuevas fuentes de conocimiento.

Las tareas que abarcaremos podría clasificarse dentro de los siguientes campos de la informática:

- **Procesamiento del lenguaje natural:** el sistema tendrá que aplicar técnicas de procesamiento de lenguaje natural para el análisis de las preguntas introducidas por el usuario y textos que contienen el conocimiento.
- **Interacción con bases de conocimiento:** el sistema deberá interactuar con bases de conocimiento que almacenan, manteniendo la información y dependencias relevantes, el conocimiento textual utilizado para responder las preguntas.
- **Recuperación de la información:** el sistema debe ser capaz de extraer información relevante partiendo de distintas fuentes de conocimiento. Estás pueden ser tanto bases de datos que almacenen información textual como sitios web que dispongan de una API (Application Programming Interface) para su acceso (12).
- **Generación de sentencias en lenguaje natural:** el sistema deberá ser capaz de generar sentencias en lenguaje natural a partir de las información obtenida por el sistema de la base y diversas fuentes de conocimiento.

1.3. Visión general del documento

El documento está dividido en varios apartados que se indicarán a continuación:

- Capítulo 2: se explica el trabajo de investigación realizado antes de la implementación del sistema.
- Capítulo 3: se explica como se ha realizado la gestión del proyecto y las herramientas utilizadas para ello.
- Capítulo 4: se desarrolla funcionamiento general del sistema, los componentes que lo forman y cómo interaccionan entre ellos para ofrecer la funcionalidad general del sistema.
- Capítulo 5: se muestra el comportamiento de nuestro sistema sobre distintos dominios de distinta complejidad lingüística y los resultados obtenidos en cada uno de ellos.
- Capítulo 6: se exponen las conclusiones alcanzadas por la realización de este proyecto y distintos aspectos o funcionalidades a implementar para mejorar el sistema en un futuro.
- Apéndice A: se explica el funcionamiento del sistema por medio de un ejemplo completo guiado.
- Apéndice B:se explican aspectos de interés para la instalación y uso del sistema implementado.

1.4. Motivation

Throughout history many question answering systems trying to satisfy human necessities have existed, such as the online customer support systems used by public transport companies. Many of these systems functioning are based on statistical studies and models trained with great volumes of data for solving the problem.

It is not always possible to have such quantity of data to train the model so many different approaches to solve the problem have appeared (8). One of the most interesting is the one based on linguistic analysis of textual knowledge.

This approach allows to solve the problem by basing on just the information contained in several texts, independiently of the domain which the texts and introduced queries belong to.

Question answering systems can be distinguished not only according to the implementation but also according to the domain the work with (11). Many systems are implemented to solve the queries related to a specific domain (a website, a repository...). Nevertheless, we can also find multiple domain systems using the great amount of information in the Internet to solve questions from different fields. These are particularly interesting because they can solve a greater range of problems.

1.5. Objectives

The main objective of this project is the construction of a question answering system capable of solving queries introduced by the user using linguistic analysis of textual knowledge.

To take advantage of the great amount and diversity of textual knowledge we can fin in the Internet, the system must be adaptable to different domains. That system must be built starting from a generic architecture (4) using different sources of knowledge. Another requirement the system must meet is to be easily integrated with new sources of knowledge.

The tasks we will be facing can be classified in the following fields of computer science:

- **Natural language processing:** the system will have to apply natural language processing techniques to analize the questions introduced by the user and the texts containing the knowledge.

- **Interaction with the knowledge base:** the system must interact with the knowledge bases containing the textual knowledge, which maintain the information and relevant dependencies.
- **Information retrievement:** the system must be capable of extracting relevant information from different sources of knowledge. These can be both databases storing textual information or websites providing an API (Application Programming Interface) (12).
- **Natural language sentences generation:** the system must be capable of generating natural language sentences from the information obtained by the system from the knowledge base and different sources of knowledge.

1.6. General vision of the document

The document si divided in the following sections:

- **Chapter 2:** explains the investigation work previous to the implementation of the system.
- **Chapter 3:** explains the project management and the tools used for that.
- **Chapter 4:** explains the general functioning of the system, its components and how they interact between them to offer the general functionality of the system.
- **Chapter 5:** shows the behavior of our system in diverse domains of different linguistic complexity and the obtained results for each of them.
- **Chapter 6:** explains the reached conclusions during the implementation of this project and the different functionalities to implement in order to improve the system.
- **Appendix A:** explains the systems functioning with a full guided example.
- **Appendix B:** provides relevant information about the instalation and usage of the implemented system.

Capítulo 2

Trabajo previo

Resumen: en este capítulo se expone el estado de la cuestión y las tecnologías investigadas para desarrollar el proyecto.

2.1. Estado del arte

Los primeros sistemas de búsqueda de respuestas[1] se empezaron a desarrollar sobre los años 60, aunque se trataba de sistemas restringidos a contextos muy concretos como BASEBALL y LUNAR, que solo respondían preguntas sobre jugadores de béisbol de USA y sobre rocas lunares, respectivamente. Ambos sistemas fueron realmente efectivos en sus dominios, siendo así que LUNAR en una convención en 1971 fue capaz de responder al 90 % de las preguntas de los usuarios.

En base a estos sistemas se empezaron a desarrollar otros más modernos pero que, al igual que los anteriores, siempre mantenían una base de conocimiento estática proporcionada por expertos en la materia. Dos claros ejemplos en relación a estos sistemas son SHRLDU y ELIZA, ambos creados entre finales de los años 60 y principio de los 70. SHRLDU era un simulador digital de bloques donde el usuario podía indicarle como mover los bloques y donde ponerlos, estableciendo una comunicación pregunta-respuesta entre ambos, mientras que ELIZA era un *bot* conversacional el cual respondía a las preguntas de los usuarios con frases almacenadas en bases de datos.

[1] Historia de los sistemas de búsqueda de respuesta: https://goo.gl/CcZmCr

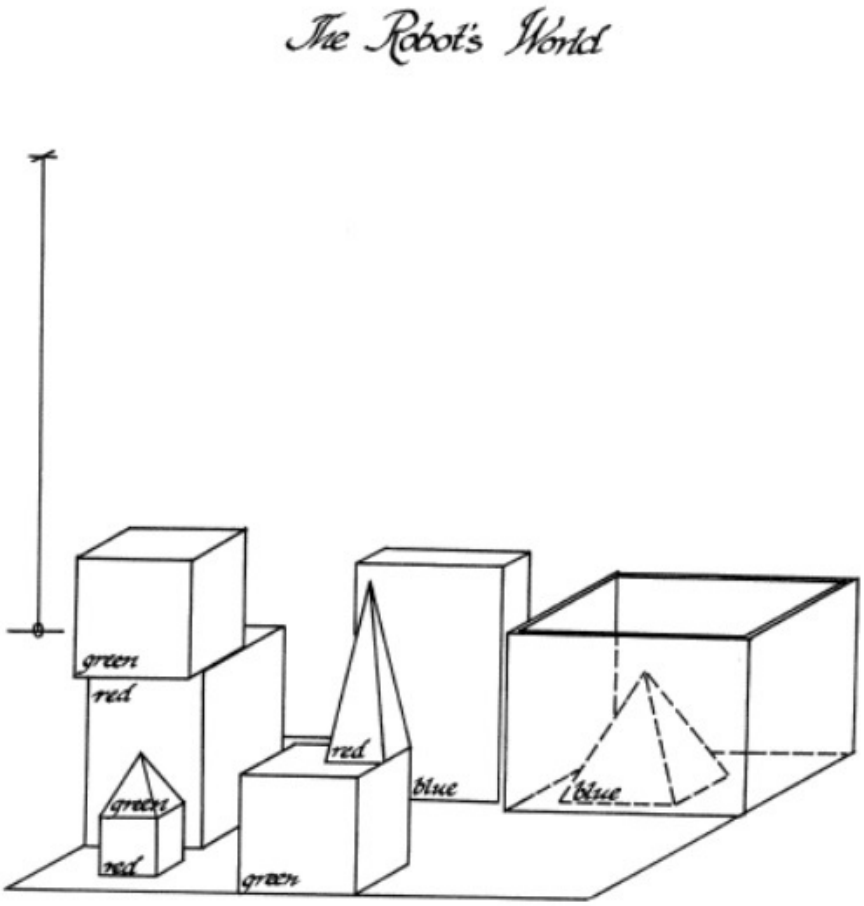


Figura 2.1: Representación del problema a resolver por el sistema SHRDLU.

Por los años 80 comienza el desarrollo de las teorías de la lingüística computacional y se empiezan a crear proyectos de QA (Question Answering) más ambiciosos que sus predecesores. El ejemplo más representativo de estos sistemas es el Unix Consultant, que respondía preguntas referentes al sistema operativo Unix pero las adecuaba en relación al tipo de usuario que las realizaba (un usuario principiante, experto, etc), aunque seguía trabajando con la misma base de conocimiento que los anteriores sistemas; estática y de dominio concreto.

La teoría de computación gramatical se convirtió en un área de investigación muy activa con el fin de conseguir un conocimiento y lógica capaces de representar las intenciones del usuario. En este marco se desarrolló el núcleo del motor lingüístico del *International's Cambridge Computer Science Research Centre(1)* y la teoría de representación del discurso. Dicho motor

lingüístico es un sistema independiente del dominio que traduce el inglés a una representación más formal. En un principio fue diseñado para ser usado como componente genérico en interfaces de bases de datos y diagnóstico de sistemas, sin embargo ayudó en gran medida al procesamiento lingüístico gracias a su arquitectura multicapa.

En la década de los 90 una de las prácticas más innovadoras fue el procesado estadístico del lenguaje, de manera que se empiezan a aplicar métodos estadísticos en cohesión con PLN (10) (Procesamiento del Lenguaje Natural), como el análisis sintáctico y semántico probabilístico. Con el crecimiento de la información en la red aparecen técnicas de procesamiento de texto, técnicas de extracción de la información y adquiere importancia la robustez y portabilidad de los sistemas, es decir, que no requieran un gran dominio del lenguaje y sean fácilmente ampliables.

En esta etapa empiezan a salir al mercado herramientas de procesado de dominio público y recursos lingüísticos como corpus y bases de datos.

2.2. Tecnologías relevantes y otras investigaciones

Para implementar el sistema ha sido necesario investigar distintas tecnologías que pudiesen ser de utilidad. En este apartado describimos las tecnologías estudiadas, y otras investigaciones, realizadas con el fin de cumplir con los objetivos del proyecto.

2.3. PostgreSQL

Structured Query Language o SQL es el lenguaje estándar utilizado en la gestión de bases de datos relacionales las cuales proporcionan atomicidad, consistencia, aislamiento y durabilidad.

Toda la información se almacena en tablas con una estructura estática, es decir, siguen el mismo formato siendo su principal ventaja la robustez a la hora de mantener la integridad de los datos. Sin embargo, esto también puede suponer un problema en cuanto a la escalabilidad del sistema en caso de que queramos modificar el modelo de dicha base de datos. Además su rendimiento disminuye considerablemente a medida que aumenta la cantidad de datos que se almacenan.

Hemos usado este tipo de bases de datos como punto de partida a la hora de desarrollar nuestro proyecto ya que disponíamos de un volcado de datos de un proyecto anterior, el cual hemos usado como fuente de conocimiento. Para poder trabajar con esta información optamos por PostgreSQL [2], un sistema de gestión de bases de datos relacionales orientada a objetos, de código abierto y bajo la licencia PostgreSQL Licence (muy permisiva y parecida a MIT).

Las principales ventajas que PostgreSQL nos ofrece van enfocadas a la velocidad con la que accedemos a la información, pues a pesar de trabajar con un gran volumen de datos y tratarse de una base de datos relacional, las consultas se ejecutan en un tiempo moderado gracias a su diseño MVCC (Multiversion Concurrency Control) que permite a los accesos de solo lectura continuar leyendo datos durante la actualización de los registros. También cabe destacar que al ser multiplataforma nos ha permitido trabajar bajo distintos sistemas operativos sin tener que cerrarnos a uno en concreto.

De cara al desarrollo del proyecto descartamos su uso bastante rápido ya que a pesar de su destacada eficiencia y de trabajar con bases de datos relacionales se nos quedaba corto a la hora de realizar búsquedas sobre un texto. Por ello, optamos por migrar los volcados a bases de datos NoSQL (Non Structured Query Language) con el objetivo de simplificar la búsqueda de respuestas a las cuestiones de los usuarios.

2.4. MongoDB

Las bases de datos NoSQL se caracterizan por ofrecer un formato mucho más flexible que las SQL al no depender de un modelo estático basado en tablas y además garantizan ACID (siglas de *Atomicity*, *Consistency*, *Isolation* y *Durability*). Esta ventaja nos proporciona una mejor visibilidad de los datos y un considerable aumento del rendimiento al poder distribuir la información sin tener que seguir un esquema fijo. Además gracias al *sharding* podemos fragmentar y distribuir los datos, siendo posible el manejo de un mayor volumen de los mismos de un modo más rápido y eficiente. MongoDB [3] nos proporciona una forma sencilla de hacerlo, pero que hay que configurar correctamente.

Para la migración desde PostgreSQL a NoSQL nos decantamos por MongoDB, uno de los principales sistemas bases de datos no relacionales. El almacenamiento de la información se realiza en documentos (por lo general JSON, JavaScript Object Notation) y destaca por su flexibilidad y escalabi-

[2] Más sobre PostgreSQL: https://www.postgresql.org/about/
[3] Más sobre MongoDB: https://www.mongodb.com/what-is-mongodb

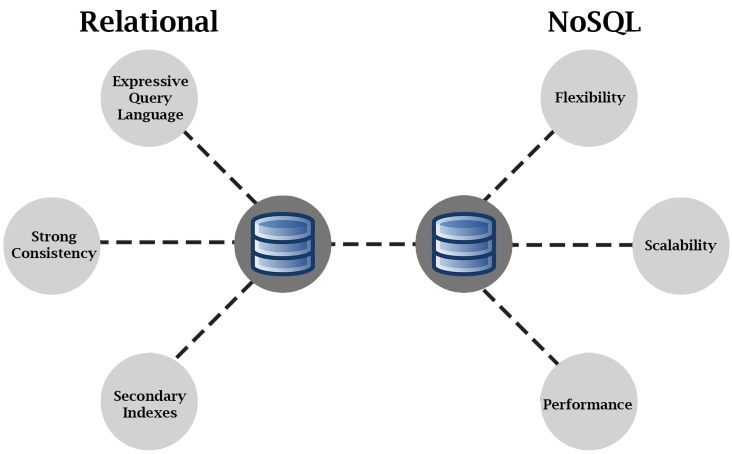


Figura 2.2: Comparativa características bases de datos SQL y NoSQL.

lidad horizontal, es decir, podemos agregar nuevos campos fácilmente cuando sea necesario sin necesidad de preocuparnos por mantener la consistencia de un modelo como ocurría con las bases de datos SQL.

MongoDB soporta *querys* para la búsqueda dentro de textos al igual que PostgreSQL pero nos proporciona una notación mucho más clara y además nos permite generar índices para mejorar las búsquedas. Estos índices se generan con una estructura tipo Árbol-B o B-Tree[4] la cual mantiene los nodos balanceados. Esto incrementa notablemente la velocidad tanto a la hora de realizar búsquedas como al devolver los resultados de las mismas ya ordenados. Además MongoDB nos permite recorrer los índices en ambos sentidos, luego con un solo índice podemos conseguir una ordenación ascendente o descendente según nos sea necesario.

Este cambio en la organización de la información nos proporcionó muchas ventajas de cara al desarrollo del sistema, principalmente en temas de rendimiento y claridad a la hora de trabajar.

2.5. Neo4J

Otro tipo de bases de datos NoSQL con las que hemos estado trabajando son las orientadas a grafos (6) pues nos proporcionan una representación más adecuada de los textos, correspondiéndose los nodos con las palabras de los mismos mientras que las aristas sirven para definir las dependencias entre dichos nodos, lo cual nos aporta mucha más información que un simple texto plano.

[4]Más sobre Árboles-B: https://es.wikipedia.org/wiki/%C3%81rbol-B

Por ese motivo nos decantamos por Neo4j[5] , una base de datos orientada a grafos *open source* desarrollada con Java. El modelo de los grafos de Neo4j consta de:

- Nodos: son los principales elementos de datos y están conectados entre sí mediante relaciones (que pueden ser múltiples para un mismo nodo e incluso recursivas). Pueden contener una o mas propiedades almacenadas como atributos clave-valor, así como varias etiquetas que describen su rol dentro del grafo.
- Relaciones: enlaces dirigidos que conectan dos nodos entre sí y poseen una o más propiedades, al igual que los nodos, almacenadas como atributos clave-valor.
- Propiedades: son cadenas de texto con valor las cuales pueden formar índices compuestos en base a varias propiedades.
- Etiquetas: se encargan de agrupar nodos en conjuntos, los cuales son indexados para encontrar otros nodos en el grafo de forma más eficiente.

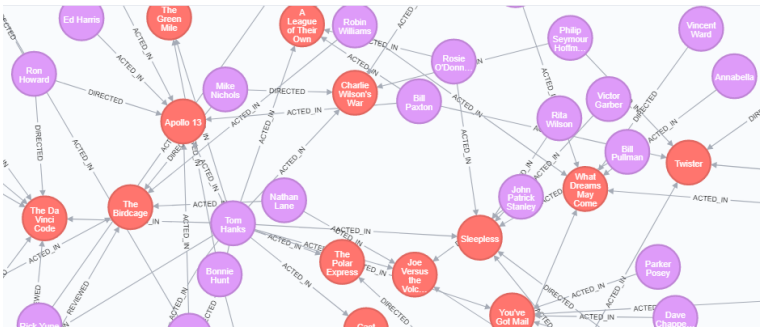


Figura 2.3: Representación del contenido de una base de datos Neo4j.

Además nos aporta grandes novedades respecto a otros sistemas de bases de datos NoSQL:

- El almacenamiento de la información como grafos con índices de adyacencia libre nos proporciona tanto transacciones como un procesamiento de datos mucho más veloz.
- El modelo de datos que propone es totalmente flexible y nos proporciona un control total de de la arquitectura de los datos.
- Las *querys* perciben una gran mejora al trabajar sobre un grafo donde las relaciones ya están creadas a diferencia de otros sistemas NoSQL donde dichas relaciones han de crearse a nivel de aplicación.

[5]Más sobre Neo4j: https://neo4j.com/why-graph-databases/

- Cuenta con Cypher[6] , un lenguaje para construir *querys* que contienen relaciones en sí mismas.
- El modelo de grafos es fácilmente escalable.

2.6. Cypher

Se trata de un lenguaje declarativo inspirado en SQL para la generación de *querys* sobre bases de datos Neo4j, el cual se caracteriza por tener una sintaxis simple, un formato de escritura muy visual y una fuerte base en la lengua inglesa, por lo que las *querys* son bastante autoexplicativas.
Cypher está construido en torno a tres elementos básicos:

- Nodos: representación de las palabras de los textos que además pueden contener varios atributos almacenados como pares clave-valor.
- Relaciones: almacenan las dependencias entre los nodos y pueden estar etiquetadas o no. Estas etiquetas pueden corresponderse con variables, propiedades adicionales e incluso información estructural sobre los *paths* que interconectan los nodos.
- Patrones: conjunto de nodos y relaciones cuyas funcionalidades van desde la relación y creación de datos y la evaluación de expresiones y resultados hasta el recorrido de árboles. También se encargan de limpiar el conocimiento acumulado innecesario, así como de la creación y filtrado tanto de nodos como aristas.

Como se muestra en la figura 2.4 las consultas en lenguaje *Cypher* están compuestas por características y relaciones que tienen los nodos de los cuales se desea extraer información. Al encontrar los nodos que encajan se devuelven las propiedades que correspondan.

```
MATCH (node:Label) RETURN node.property

MATCH (node1:Label1)-->(node2:Label2)
WHERE node1.propertyA = {value}
RETURN node2.propertyA, node2.propertyB
```

Figura 2.4: Ejemplo de sentencia en lenguaje Cypher.

[6]Más sobre Cypher: https://en.wikipedia.org/wiki/Cypher_Query_Language

2.7. Python

Python[7] es un lenguaje multiplataforma, interpretado y con tipado dinámico que además soporta orientación a objetos, programación imperativa y también, aunque en menor medida, programación funcional.

Uno de los principales motivos por el que desarrollamos el proyecto en torno a este lenguaje fueron la gran cantidad de bibliotecas y documentación de las que dispone, además de tratarse de un lenguaje con una sintaxis simple y una buena curva de aprendizaje.

La construcción del proyecto se ha realizado a partir de ficheros .py[8] y notebooks[9] . Estos últimos nos proporcionaron una forma sencilla y muy visual de testear las implementaciones que íbamos desarrollando, sin embargo su uso suponía una gran carga para el entorno de trabajo luego tuvimos que reducirlos en gran medida, quedándonos solamente con aquellos notebooks imprescindibles para testeo. Para aliviar este problema en lo máximo posible comenzamos por migrar los notebook a ficheros .py ya que eran mucho más ligeros, lo cual implicó adaptar y reestructurar la mayor parte del código ya implementado.

2.8. NLTK

NLTK (3) es sin duda alguna la herramienta de referencia (gratuita y open source) en lo que respecta a PLN en Python desde hace bastantes años.

No solo nos ofrece una amplia cantidad de algoritmos sino que además nos permite personalizar los propios en gran medida y es por ello que está especialmente centrado en campos cómo la enseñanza o la investigación. Además ha sido una librería capaz de resolver muchos problemas de análisis de texto en primera instancia.

No obstante, pese a ser uno de los mayores referentes en este campo presenta varias carencias muy importantes. Como ya comentamos antes nos permite una amplia personalización de nuestros algoritmos, sin embargo esto conlleva una curva de aprendizaje demasiado elevada como para conseguir un dominio de la herramienta en un tiempo razonable. Además, está el hecho de que existen librerías como SpaCy[10] que ya nos proporcionan procesamientos óptimos para muchas situaciones, luego carece de sentido invertir tiempo y

[7] Más sobre Python: https://www.python.org/about/
[8] ficheros propios de Python
[9] ficheros de Jupyter Notebook: http://jupyter.org/
[10] Más sobre SpaCy: https://spacy.io/

esfuerzo de forma innecesaria.

Otra de las desventajas a resaltar es su eficiencia, pues las velocidades de respuesta que presenta son bastante peores que las de otras librerías, luego en base a estos dos puntos descartamos NLTK[11] como opción viable para resolver la cuestión.

2.9. TextBlob

Esta herramienta tiene una ventaja considerable respecto al resto de librerías Python para PLN; su simplicidad de uso. Presenta menos funcionalidades que otras librerías como NLTK pero su facilidad de uso permite explotarlas mejor con un conocimiento menor de la herramienta.

TextBlob[12] realiza tareas tales como traducción de textos, clasificación, extracción de palabras "relevantes", etiquetado ("tagging") de las distintas palabras de una oración o "lematización" (extracción de la raíz de un término), tareas básicas que necesitamos para el desarrollo del proyecto.

Después de realizar varias pruebas de concepto con esta tecnología, decidimos no utilizarla en nuestro sistema dado que hay otras herramientas que, a pesar de ser más complejas de aprender a usar, nos proporcionan mejores resultados.

2.10. SpaCy

SpaCy surgió en el año 2015 como una potente herramienta gratuita y *open source* de PLN para Python, la cual está considerada una de las mejores en este campo a día de hoy. Además se centra en proveer software para uso real y no tan teórico como NLTK (15).

Hasta el momento, esta herramienta proporciona soporte para un total de 7 idiomas distintos, aunque los mejores resultados se obtienen cuando se procesan sentencias en inglés.

Como principales ventajas tenemos un analizador sintáctico muy rápido y preciso (el mejor entre las principales librerías de PLN hasta el día de hoy) y una filosofía orientada a resolver el problema, sin hacer hincapié en aspectos teóricos innecesarios.

[11] Más sobre NLTK: https://www.nltk.org/
[12] Más sobre TextBlob: http://textblob.readthedocs.io/en/dev/

Algunas de las principales funcionalidades que hacen atractivo a Spacy son la obtención de términos relevantes o la generación de árboles de análisis sintáctico de sentencias en lenguaje natural, como se muestra en la siguiente figura:

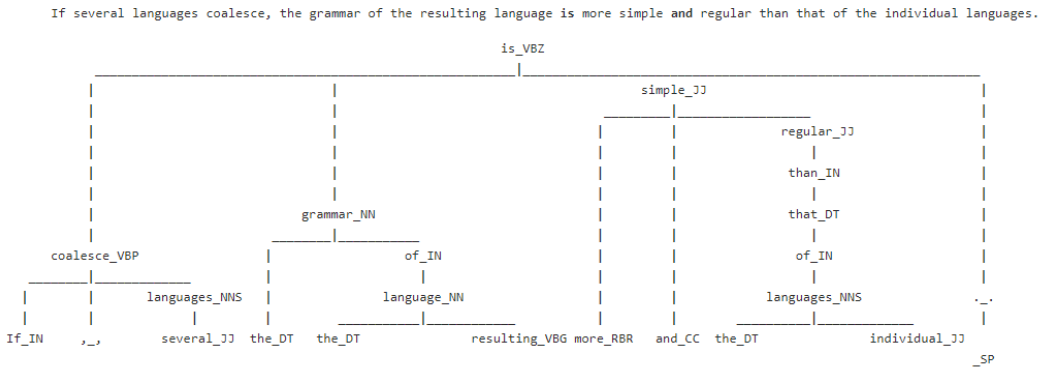


Figura 2.5: Árbol de análisis sintáctico generado por Spacy.

Aún así no todo son ventajas, pues la instalación de la librería se hace compleja (especialmente en equipos con Windows) al tener que instalar primero el núcleo de la misma y a continuación los paquetes de idiomas que han de conectarse al mismo. Además, al tratarse de una librería relativamente nueva nos encontramos con una comunidad reducida, luego no es fácil encontrar solución a según que problemas al tener pocas fuentes de información y tratarse de una librería que cubre unas necesidades muy específicas.

Debido esto decidimos realizar varias pruebas de concepto con SpaCy y los resultados fueron realmente satisfactorios. Con relativa facilidad nos permitía llevar a cabo tareas bastante complejas como detectar frases, etiquetar palabras con categorías gramaticales u obtener palabras que pertenecen al mismo campo semántico (siendo esta última considerablemente más compleja).

En base a estos resultados nos decidimos por esta herramienta aunque no la aplicamos de manera directa como se hizo en las pruebas de concepto, sino que Spacy actúa como el núcleo dentro de nuestra herramienta principal, Grafeno (más adelante profundizaremos en el funcionamiento conjunto de ambas librerías).

2.11. Autocorrect

Autocorrect[13] es una librería escrita en Python que realiza correcciones ortográficas de textos en inglés en lenguaje natural.

Para llevar a cabo la corrección ortográfica genera una lista de posibles candidatos para cada término a corregir. A continuación comprueba si la palabra original tiene algún error ortográfico y en caso de ser así lo substituye por el candidato con mayor probabilidad de ser usado.

Dada la naturaleza de la implementación que posee solo es capaz de corregir ortográficamente término a término, luego es incapaz de detectar errores asociados a concordancia de tiempo verbales, de género o número.

2.12. Grafeno

Grafeno[14] es una librería Python creada por Antonio F. García Sevilla y Alberto Díaz con licencia AGPL-3.

Permite construir grafos semánticos a partir de árboles de análisis sintáctico generados por otras tecnologías como Spacy o Freeling[15] . También tiene la capacidad de realizar el proceso descrito anteriormente a la inversa, es decir, generar sentencias en lenguaje natural a partir de grafos semánticos.

Otra de sus principales funcionalidades es la de crear *querys* de distintos tipos en lenguaje Cypher a partir de estos grafo, como *querys* de consulta o de inserción de información para una base de datos Neo4J.

Para entender su funcionamiento, es necesario destacar cuáles son los elementos que componen Grafeno:

- Transformadores: componentes que se utilizan en la generación del grafo semántico a partir de un árbol de análisis sintáctico. Cada transformador se encarga de un aspecto de dicho árbol, como pueden ser las partículas interrogativas o los sustantivos que contiene.

[13] Más sobre Autocorrect: https://pypi.org/project/autocorrect/
[14] Más sobre Grafeno: https://github.com/agarsev/grafeno
[15] Más sobre Freeling: http://nlp.lsi.upc.edu/freeling/index.php/

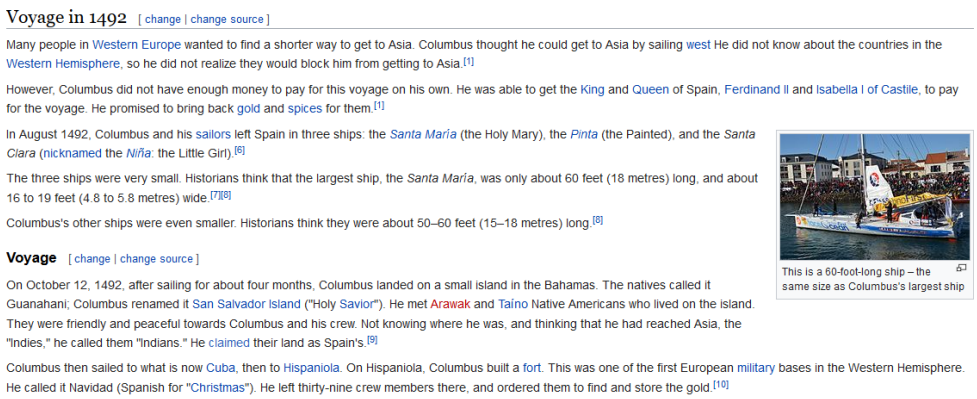
- Linearizadores: elementos utilizados para la generación de sentencias, tanto en lenguaje natural como en Cypher, a partir de un grafo semántico. Cada uno de ellos trata un aspecto del grafo semántico para la "traducción" de éste a una expresión. En función del tipo de expresión que se desee generar se utilizarán unos linearizadores u otros.

Para la implementación del sistema se ha utilizado Grafeno a la vez que se han realizado una serie de modificaciones para añadir funcionalidades interesantes para nuestro sistema. En apartados posteriores se comentarán cada una de las modificaciones añadidas.

2.13. Simple Wikipedia

Simple Wikipedia[16] es una enciclopedia online basada en su versión más famosa (Wikipedia) pero utilizando un lenguaje y gramática más sencillos que la Wikipedia estándar, utilizando por ejemplo frases más cortas y palabras más comunes. El objetivo principal de esta Wikipedia es hacer la misma más accesible a personas con dificultades, dominio reducido del idioma o dificultades de aprendizaje.

Actualmente en Simple Wikipedia hay mucha menos información e idiomas posibles que en su versión principal, no obstante, en el ámbito de procesamiento de lenguaje natural nos ofrece una gran cantidad de posibilidades debido a que facilita la tarea del análisis sintáctico y semántico de la información contenida en la misma.



Voyage in 1492 [change | change source]

Many people in Western Europe wanted to find a shorter way to get to Asia. Columbus thought he could get to Asia by sailing west He did not know about the countries in the Western Hemisphere, so he did not realize they would block him from getting to Asia.[1]

However, Columbus did not have enough money to pay for this voyage on his own. He was able to get the King and Queen of Spain, Ferdinand II and Isabella I of Castile, to pay for the voyage. He promised to bring back gold and spices for them.[1]

In August 1492, Columbus and his sailors left Spain in three ships: the *Santa María* (the Holy Mary), the *Pinta* (the Painted), and the *Santa Clara* (nicknamed the *Niña*: the Little Girl).[6]

The three ships were very small. Historians think that the largest ship, the *Santa María*, was only about 60 feet (18 metres) long, and about 16 to 19 feet (4.8 to 5.8 metres) wide.[7][8]

Columbus's other ships were even smaller. Historians think they were about 50–60 feet (15–18 metres) long.[6]

This is a 60-foot-long ship – the same size as Columbus's largest ship

Voyage [change | change source]

On October 12, 1492, after sailing for about four months, Columbus landed on a small island in the Bahamas. The natives called it Guanahani; Columbus renamed it San Salvador Island ("Holy Savior"). He met Arawak and Taíno Native Americans who lived on the island. They were friendly and peaceful towards Columbus and his crew. Not knowing where he was, and thinking that he had reached Asia, the "Indies," he called them "Indians." He claimed their land as Spain's.[9]

Columbus then sailed to what is now Cuba, then to Hispaniola. On Hispaniola, Columbus built a fort. This was one of the first European military bases in the Western Hemisphere. He called it Navidad (Spanish for "Christmas"). He left thirty-nine crew members there, and ordered them to find and store the gold.[10]

Figura 2.6: Artículo perteneciente a Simple Wikipedia.

[16] Más sobre Simple Wikipedia: https://simple.wikipedia.org/wiki/Main_Page

First voyage

On the evening of 3 August 1492, Columbus departed from Palos de la Frontera with three ships: a larger carrack, the *Santa María* ex-*Gallega* ("Galician"), and two smaller caravels, the *Pinta* ("The Pint", "The Look", or "The Spotted One") and the *Santa Clara*, nicknamed the *Niña* ("Girl") after her owner Juan Niño of Moguer.[48] The monarchs forced the citizens of Palos to contribute to the expedition. The *Santa María* was owned by Juan de la Cosa and captained by Columbus. The *Pinta* and the *Niña* were piloted by the Pinzón brothers (Martín Alonso and Vicente Yáñez).[32]

Columbus first sailed to the Canary Islands, which belonged to Castile. He restocked provisions and made repairs in Gran Canaria, then departed from San Sebastián de La Gomera on 6 September, for what turned out to be a five-week voyage across the ocean. At about 2:00 in the morning of 12 October (21 October, Gregorian Calendar New Style), a lookout on the *Pinta*, Rodrigo de Triana (also known as Juan Rodríguez Bermeo), spotted land, and immediately alerted the rest of the crew with a shout. Thereupon, the captain of the *Pinta*, Martín Alonso Pinzón, verified the discovery and alerted Columbus by firing a lombard.[49] Columbus later maintained that he himself had already seen a light on the land a few hours earlier, thereby claiming for himself the lifetime pension promised by Ferdinand and Isabella to the first person to sight land.[32][50]

Figura 2.7: Artículo perteneciente a la Wikipedia estándar.

2.14. BioAsq Challenge

BioAsq Challenge[17] es un desafío organizado por miembros de distintas universidades, tanto europeas como de USA, que propone distintos retos relacionados con diversas ramas del procesamiento de lenguaje natural, tratamiento de textos y, en general, de la inteligencia artificial en el ámbito biomédico.

Los objetivos principales de este desafío son la resolución de los siguientes problemas:

- Indexación biomédica online a gran escala (16): en esta tarea, se les pide a los participantes que clasifiquen nuevos documentos de PubMed[18] (motor de búsqueda de libre acceso a la base de datos de Medline[19]) antes de que los médicos los clasifiquen manualmente. A medida que se realizan estas clasificaciones manuales, se evalúa el rendimiento de clasificación de los sistemas participantes.
- Sistema de respuesta a cuestiones biomédicas en lenguaje natural: esta tarea utiliza conjuntos de testeo, en inglés, junto con respuestas "ideales" construidas por un equipo de expertos biomédicos. Los participantes tienen que responder con información relevante, artículos y fragmentos, así como construir su semántica formal de acuerdo a una sintaxis concreta RDF[20].

[17] Más sobre Bioasq Challenge: http://bioasq.org/about
[18] Más sobre Pubmed: https://www.ncbi.nlm.nih.gov/pubmed/
[19] servicio de información en línea provisto por la Biblioteca Nacional de Medicina de los Estados Unidos
[20] lenguaje para especificar metadatos

Capítulo 3

Metodología de trabajo

Resumen: en este capítulo se expone la gestión del proyecto explicando las metodologías de trabajo aplicadas y las herramientas utilizadas, así como los problemas que tuvimos con ellas durante el desarrollo.

3.1. Introducción

La metodología de trabajo aplicada al principio del desarrollo dista mucho de la empleada al final del mismo.

Al comienzo del desarrollo optamos por realizar labores de investigación con el fin de sentar las bases del proyecto. Para ello comenzamos dividiendo el proyecto en dos ramas: recuperación de información y procesamiento de lenguaje natural. En base a esta distinción optamos por dividirnos en parejas de forma que pudiésemos trabajar en paralelo y de forma independiente.

En cuanto a la parte de recuperación de información nos centramos en buscar posibles fuentes de las que extraer información, como bases de datos o recursos en Internet como Wikipedia[1], además de probar tecnologías para determinar aquellas que mejor se ajustasen a nuestras necesidades, mientras que en la rama de procesamiento de lenguaje natural nos centramos en probar tecnologías que podrían ser útiles en un futuro. Para ello desarrollamos varias pruebas de concepto donde tratábamos de analizar textos y obtener la mayor información de ellos.

[1] Más sobre Wikipedia: https://es.wikipedia.org/wiki/Wikipedia:Portada

Esta *primera fase del desarrollo* nos ocupó cerca del primer cuatrimestre del curso y obtuvimos resultados bastante buenos respecto a las pruebas realizadas. Llegados a este punto nuestros tutores nos recomendaron usar Grafeno como base para el desarrollo y construir a partir de él el sistema de búsqueda de respuestas.

A partir de aquí cambiamos completamente la mecánica de trabajo respecto a lo establecido anteriormente. Pasamos a desarrollar el proyecto en base a *issues* con tareas específicas para trabajar de forma más eficiente.

Además empezamos a usar herramientas como Gitlab[2] , Jupyterlab[3] y Mattermost[4] , las cuales nos permitían gestionar y comunicarnos de manera eficiente.

3.2. Mattermost

Usamos esta herramienta de cara a la gestión general del proyecto. Para ello estuvimos comunicándonos a través de 3 canales en función de nuestras necesidades:

- **Consultas**: en este canal dejábamos todas las dudas en relación a las *issues*, así como los errores y problemas que fuimos obteniendo con las mismas.
- **General**: este canal era usado por los tutores como vía de comunicación para avisarnos sobre nuevas *issues*, modificaciones sobre los proyectos existentes o caídas del servidor.
- **Gestión**: este canal se usaba para concretar reuniones semanales para revisar el estado del desarrollo.

[2]Más sobre GitLab: https://about.gitlab.com/
[3]Más sobre JupyterLab: https://github.com/jupyterlab/jupyterlab
[4]Más sobre Mattermost: https://about.mattermost.com/

3.3. GitLab

Usamos GitLab como servicio de control de versiones para nuestro proyecto. Para ello estuvimos trabajando usando *issues* propuestas por los tutores, las cuales eran tareas cortas donde se indicaban los puntos sobre los que trabajar para llevar a cabo dicha tarea.

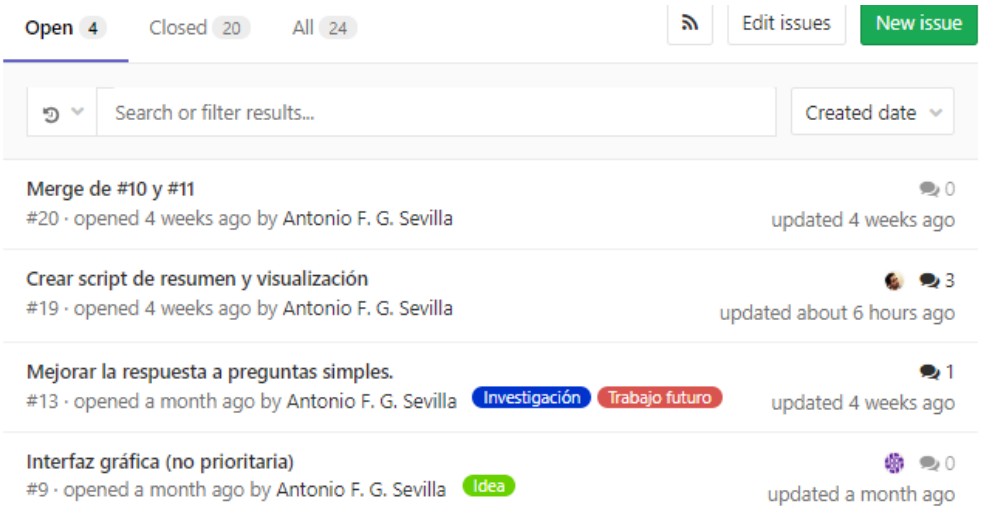


Figura 3.1: Ejemplo de gestión y distribución de *issues*.

Para empezar a trabajar en dichas *issues* creábamos una rama de trabajo desde JupyterLab (una por *issue*) de manera que la íbamos actualizando poco a poco con las nuevas modificaciones introducidas. Una vez terminábamos la *issue* en la que estuviésemos trabajando se la asignábamos a nuestros tutores para que pasasen a validarlas, y posteriormente las incluían en la rama *master* del proyecto.

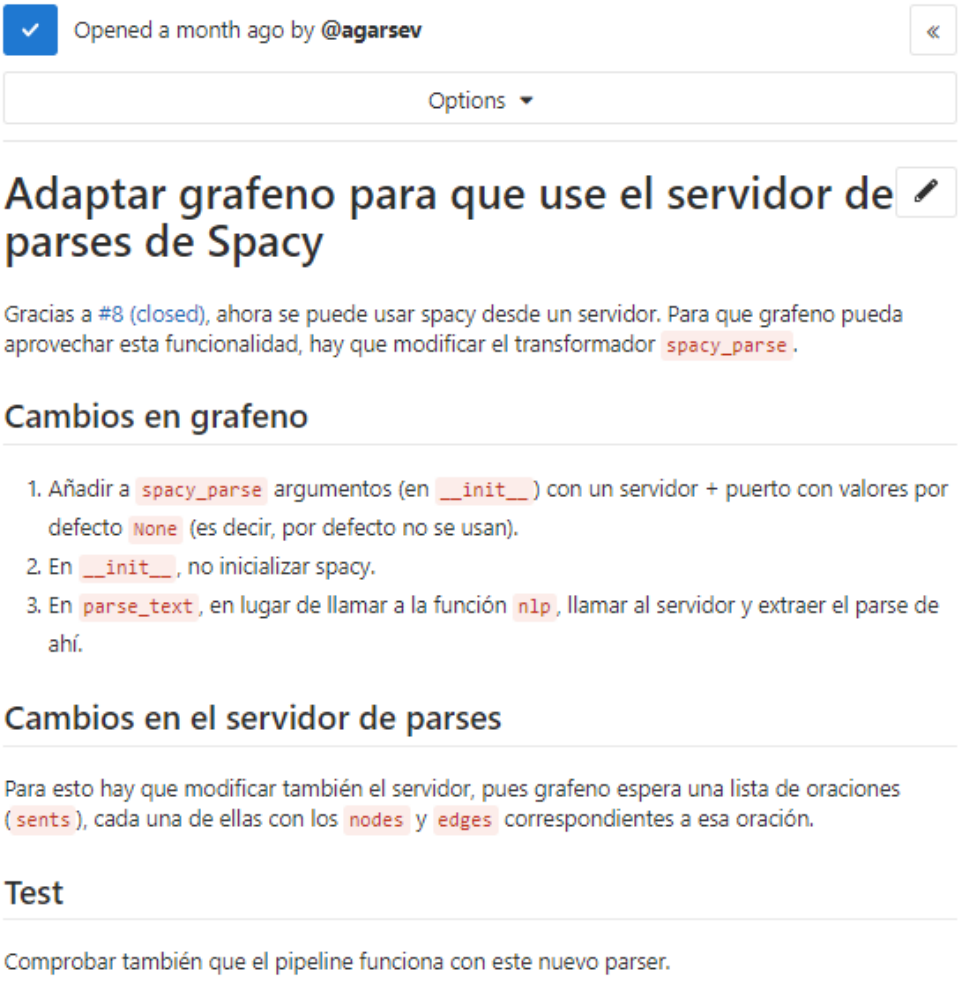


Figura 3.2: Ejemplo de *issue* a resolver.

3.4. JupyterLab

En primera instancia pensamos trabajar en local por mayor comodidad ya que no dependíamos de un servicio externo para ello, pero esta idea se quedó en el tintero dado que nos fue bastante difícil crear un entorno de trabajo completamente funcional debido a la gran cantidad de librerías que necesitábamos, pues teníamos grandes problemas de compatibilidades tanto en sistemas Windows como Unix. Estos problemas venían asociados, en general, a Spacy y a los paquetes que necesitaba para funcionar, lo cual cortó de inmediato esta vía pues esta librería era uno de los *pilares* de Grafeno, luego en vista de la situación pasamos a trabajar en JupyterLab.

JupyterLab nos ofrecía un entorno online donde trabajar cómodamente utilizando Jupyter Notebooks, una herramienta bastante útil y visual a la hora de realizar pruebas. Además, para aligerar la ejecución de la herramienta, optamos por migrar los notebooks que no se ejecutaban individualmente a ficheros Python, los cuales requerían menos recursos de la herramienta.

Capítulo 4

Sistema de búsqueda de respuestas adaptable a distintos dominios

Resumen: en este capítulo se expone una visión general del sistema implementado y de sus componentes, explicando las relaciones existentes entre ellos para formar el sistema.

4.1. Funcionamiento general del sistema

Nuestro sistema de búsqueda de respuestas analiza, mediante técnicas de procesamiento de lenguaje natural, la consulta introducida por el usuario para obtener aquellos aspectos que pueden ser relevantes de cara a la posterior búsqueda de respuestas, como las partículas interrogativas que determinan el tipo de pregunta o los términos que complementan a sustantivos o verbos (en este caso adjetivos y adverbios respectivamente).

El resultado de este procesamiento se utiliza para obtener información relevante usada en la generación de la respuesta, de forma que al terminar el análisis de la consulta buscamos en la base de conocimiento documentos relacionados. Éstos se buscan evaluando sentencias similares al resultado del análisis de la consulta dentro de la base de conocimiento relevante, estudiando si estas expresiones responden verdaderamente a la cuestión planteada.

Es importante matizar que los datos almacenados en la base de conocimiento relevante se obtienen de diferentes fuentes en función de la configuración del sistema, de manera que cuando se realizan consultas a la base de conocimiento actual y ésta no las puede responder, se realiza una petición a otra fuente en función de la configuración del sistema y la respuesta obtenida de la misma se almacena en la base de conocimiento relevante para ser reutilizada posteriormente.

Por último, a partir de las respuestas proporcionadas se generan expresiones en lenguaje natural que contienen toda la información relevante extraída con el objetivo de responder correctamente a la consulta introducida por el usuario (2).

En la figura 4.1 se muestra un diagrama que representa el funcionamiento general del sistema, pasando por cada uno de los pasos descritos en los párrafos anteriores. Por otro lado, en la figura 4.2 se muestra un ejemplo concreto (de alto nivel) del funcionamiento general del sistema.

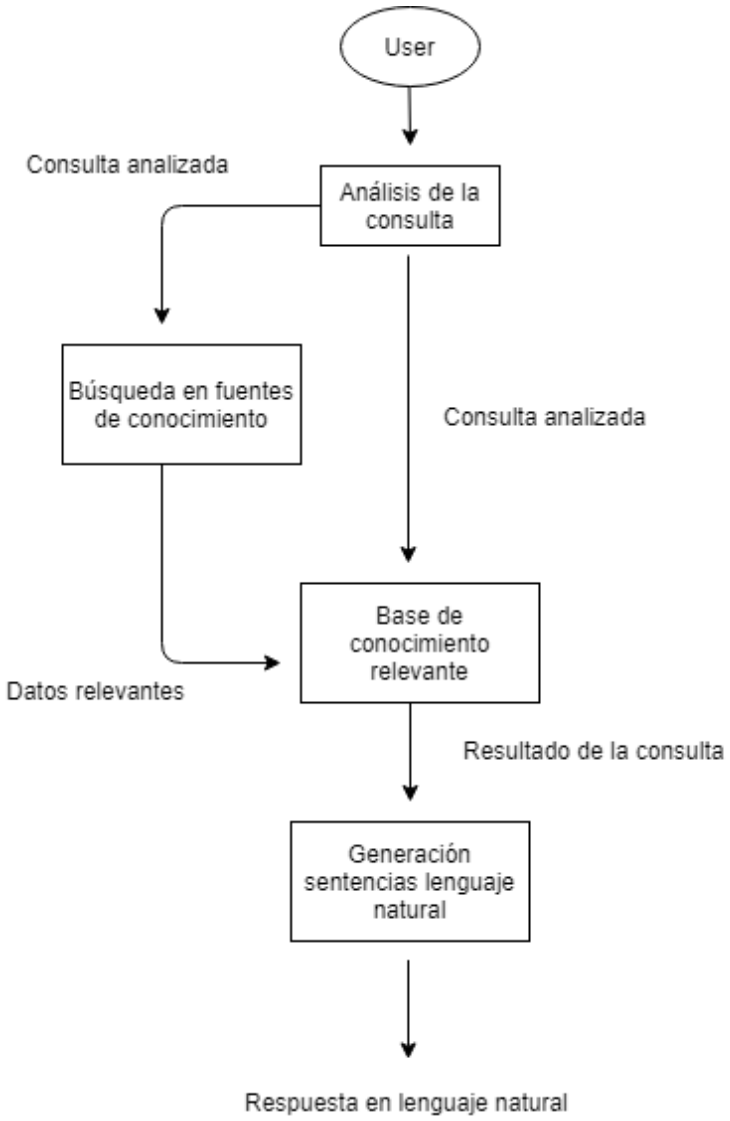


Figura 4.1: Diagrama que representa el funcionamiento general del sistema.

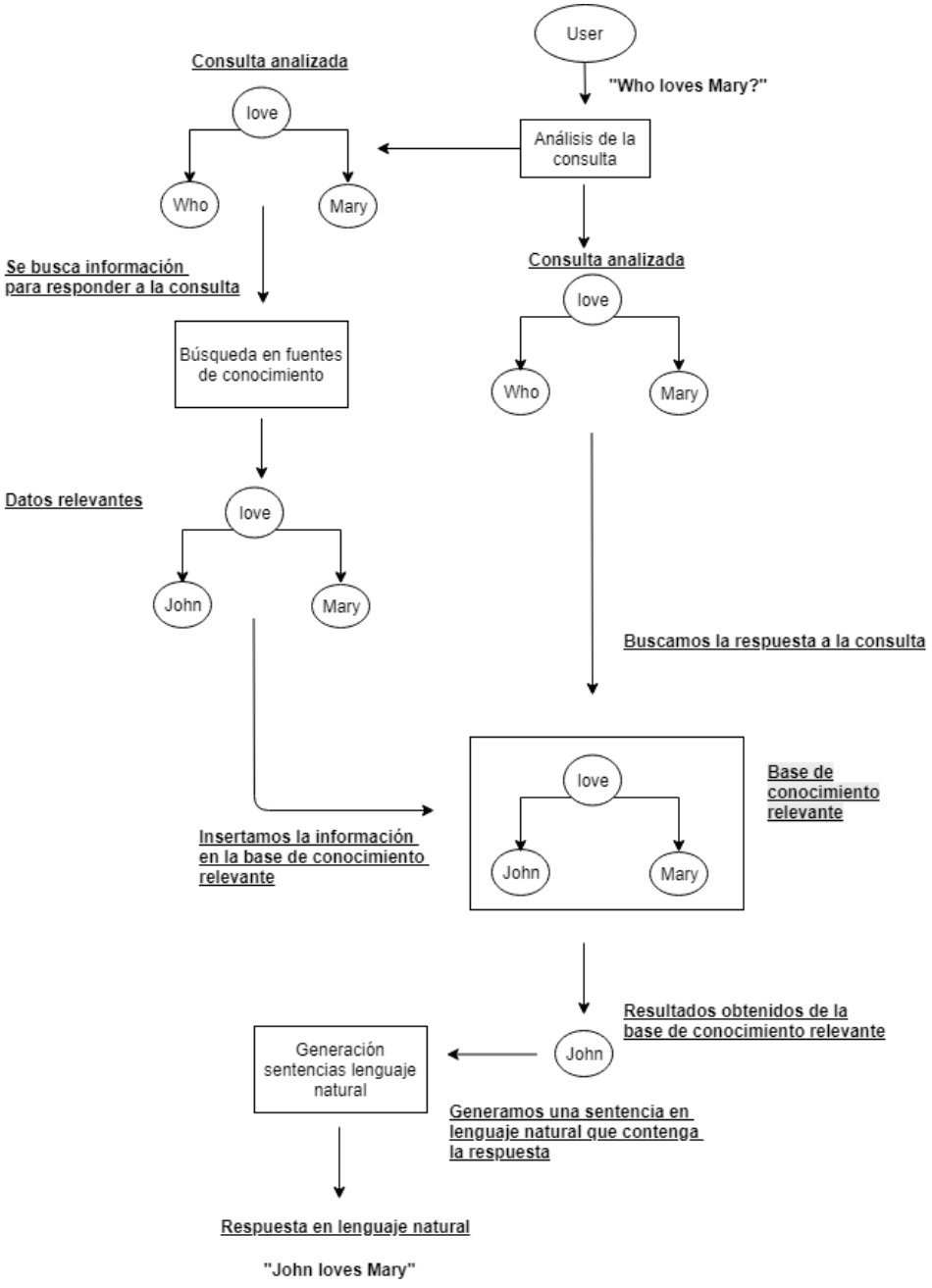


Figura 4.2: Ejemplo de alto nivel de funcionamiento general del sistema

4.2. Resolución de problemas en múltiples dominios

Tal y como se comentaba en el apartado anterior, es importante distinguir entre las fuentes de datos y la base de conocimiento relevante, o simplemente base de conocimiento. La base de conocimiento relevante es la encargada de almacenar toda la información que utilizará nuestro sistema para responder a consultas introducidas por el usuario, mientras que las fuentes de datos son aquellos recursos que usamos en caso de que la información contenida en la base de conocimiento no sea suficiente para la resolver una consulta.

Normalmente el término ”base de conocimiento” engloba todo el conocimiento que utiliza el sistema para responder a las peticiones o consultas del usuario. Nosotros, para diferenciar entre la información almacenada en el sistema y los lugares de los cuales obtiene más información, hacemos la distinción entre base de conocimiento relevante y fuentes de datos/información.

A pesar de que nuestro sistema utiliza una única base de conocimiento, el programa está implementado de forma que pueda utilizar distintas fuentes de información según sea necesario. Además se ha diseñado de tal manera que permite integrar distintas fuentes de datos independientemente de la arquitectura del sistema, siempre que dichos datos sean información textual.

La posibilidad de trabajar sobre distintas fuentes de datos dota al sistema de una capacidad de resolver problemas de distinta índole, aunque el dominio de trabajo dependerá exclusivamente del contenido de dichas fuentes de información.

Actualmente, disponemos de dos fuentes de información:

- **Simple Wikipedia:** obtiene los datos mediante la *API* pública de Simple Wikipedia. Los documentos obtenidos de esta fuente se caracterizan por tener un lenguaje fácil de procesar y analizar al usar una sintaxis y semántica simplificada respecto a la Wikipedia estándar.
- **Artículos médicos:** conjunto de artículos de ámbito médico de proyectos anteriores. Estos artículos se caracterizan por tener un lenguaje más específico que los textos de Simple Wikipedia y, en muchos casos, una sintaxis y semántica más compleja.

Esta información se ha volcado tanto en una base de datos SQL (accediendo con PostgreSQL) como en una NoSQL (MongoDB) con la finalidad de comparar el rendimiento de ambos tipos de bases de datos y escoger el que mejores resultados proporcionase. El escogido finalmente debido a los resultados obtenidos fue el volcado a MongoDB.

Para realizar pruebas con el fin de comprobar la capacidad de resolución de nuestro sistema hemos utilizado ficheros de *testing* de Simple Wikipedia y BioASQ Challenge. El resultado de estas pruebas se encuentran en apartados posteriores.

4.3. Componentes del sistema

Nuestro sistema está formado por varios componentes de forma que uno o varios de ellos implementan los pasos necesarios para llevar a cabo la búsqueda de respuestas a las consultas introducidas por el usuario en lenguaje natural (8).

Antes de proceder a explicar de forma más detallada cada componente por separado, vamos a introducirlos brevemente para poder ver más adelante como interactúan entre sí:

- **Grafeno:** esta herramienta es la encargada de asociar las consultas en lenguaje natural con peticiones a la base de conocimiento.
- **Módulo de obtención de datos para la base de conocimiento:** este componente es el encargado de ampliar la información contenida en la base de conocimiento a la que se le realizarán las consultas para generar una respuesta adecuada. Debido a su implementación, nos permite obtener datos de distintas fuentes en función de la configuración del sistema, independientemente de su dominio.
- **Módulo de análisis:** se encarga del análisis sintáctico y semántico (entre otras funciones) de las sentencias en lenguaje natural, tanto las introducidas por el usuario como las obtenidas de las distintas fuentes de conocimiento utilizando el módulo de obtención de datos comentado anteriormente.
- **Base de conocimiento relevante:** es la responsable de almacenar todo el conocimiento que utilizará nuestro sistema para generar respuestas adecuadas.

- **Módulo de interacción con la base de conocimiento:** es el encargado de gestionar la interacción con la base de conocimiento. Actúa como intermediario entre cualquier otro componente del sistema y la base de conocimiento.
- **Módulo controlador:** este modulo no proporciona ninguna funcionalidad como tal, sin embargo es el encargado de coordinar los componentes descritos anteriormente para hacer funcionar el sistema (a diferencia del resto de los componentes, en los siguientes apartados no se procederá a una explicación más detallada de este módulo ya que tan solo une los otros realizando peticiones a sus funcionalidades principales).

En el siguiente diagrama (Figura 4.3) se muestra como estos componentes interactúan entre sí en una resolución concreta. Se detallarán, en los siguientes apartados, los comportamientos de cada uno de estos módulos y de las tecnologías que los componen.

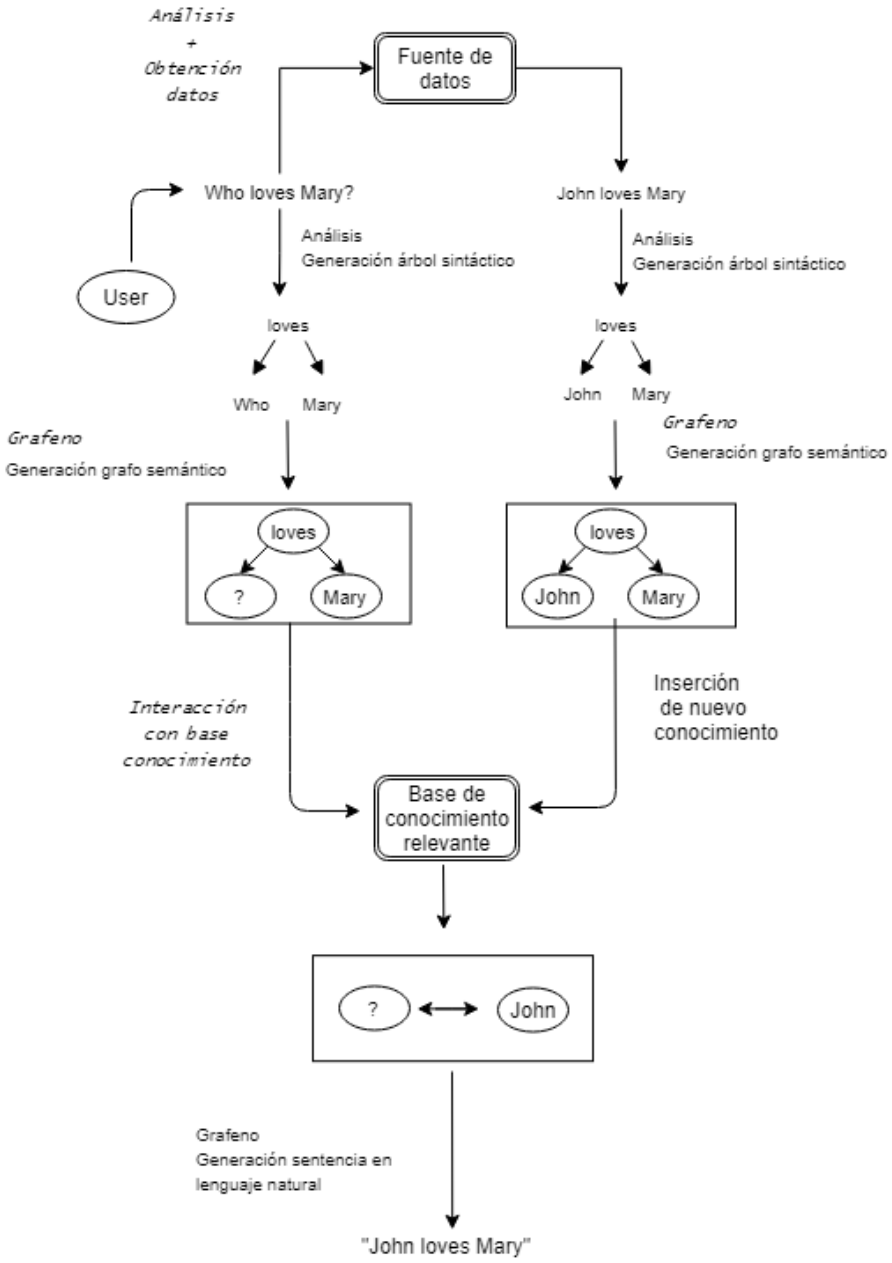


Figura 4.3: Interacción de los componentes del sistema para una ejecución concreta .

4.3.1. Grafeno

Este es uno de los componentes más importantes de todo el sistema ya que sus funcionalidades son las que permiten que otros módulos con papeles clave en el funcionamiento del sistema puedan realizar sus tareas.

Esta librería Python se utiliza para realizar tres funcionalidades principales:

- **Generación de grafos semánticos:** genera grafos semánticos (representación de conocimiento lingüístico y sentencias en lenguaje natural en forma de grafo dirigido) asociados a un texto. Utilizamos esta funcionalidad, por ejemplo, para generar el grafo semántico que representa la información que se almacenará en la base de conocimiento. La construcción de dicho grafo permite utilizar el conocimiento textual con el que trabaja el sistema de una forma más sencilla y eficiente, pero conservando las relaciones y dependencias existentes en los propios textos.
- **Generación de sentencias en lenguaje natural:** dado un grafo semántico, esta herramienta construye oraciones en lenguaje natural a partir de él preservando en ellas las dependencias y relaciones encontradas en el mismo. En concreto usamos esta funcionalidad para generar dichas expresiones en lenguaje natural a partir de las respuestas obtenidas de la consulta a la base de conocimiento. Para lograr esto se hace uso de la librería SimpleNLG[1] la cuál a veces genera frases no muy naturales, no obstante, esto no es importante puesto que lo que más nos interesa es que la respuesta sea la correcta, no que esté perfectamente formulada.
- **Generación de sentencias en lenguaje Cypher:** a partir de un grafo semántico Grafeno puede generar consultas en lenguaje Cypher. Recordemos que Cypher es el lenguaje que se utiliza para interactuar con bases de datos de Neo4J (orientadas a grafos). Usamos esta funcionalidad para:
  - Generar *querys* para incluir información en la base de conocimiento a partir del grafo semántico generado.
  - Generar *querys* de consulta a la base de conocimiento relevante utilizando el grafo semántico generado a partir de la propia consulta del usuario.

Además, sobre la librería original se han realizado modificaciones para añadir funcionalidades interesantes a nuestro sistema. Estas son:

[1] SimpleNLG: https://github.com/simplenlg/simplenlg

- **Diferenciación entre conversaciones en las sentencias Cypher:** hemos modificado Grafeno para poder almacenar y modificar contenido referente a distintas conversaciones. A cada conversación existente en el sistema se le asocia la información obtenida (y almacenada en la base de conocimiento) para resolver sus consultas. Las demás sesiones no tendrán acceso a esa información sino que cada sesión sólo tendrá acceso al conocimiento asociado a ella.

- **Separación explícita de preguntas abiertas y cerradas (13):** inicialmente Grafeno presentaba una diferenciación implícita entre preguntas abiertas y cerradas. Dada esta implementación, los módulos ajenos a Grafeno o aquellos que lo utilizaban no tenían noción del tipo de consulta con la que estaban tratando. Para poder procesar de forma distinta cada tipo de pregunta desde el resto del sistema hemos modificamos dicha librería para que diferenciase entre estos dos tipos de preguntas. Para ello implementamos varios linearizadores, uno para cada tipo de pregunta, haciendo que cada módulo que use Grafeno pueda aplicar procesamientos distintos en función del tipo de pregunta introducida por el usuario.

- **Uso de un servidor remoto para las tareas de Spacy:** para aumentar el rendimiento del sistema implementamos un servidor remoto que realizaba las tareas referentes a Spacy. De este modo aligeramos la carga en el procesamiento que debe realizar nuestro sistema, cediéndola a otro computador para que lo ejecute en paralelo. Además al usar este esquema estamos evitando la carga innecesaria de modelos Spacy.

4.3.2. Módulo de análisis

Este módulo es el responsable de realizar el análisis sintáctico y el análisis semántico de toda entrada, ya sea la consulta introducida por el usuario o los documentos obtenidos durante la fase de recuperación de información, los cuales tras ser analizados se insertarán en la base de conocimiento relevante. También realiza otras tareas como obtener similitudes sintácticas entre textos, correcciones ortográficas o determinar palabras relevantes dentro de las consultas de los usuarios (Figura 4.4) para poder proporcionar criterios de búsqueda al módulo de obtención de datos.

Respecto del análisis sintáctico, como se comentó antes, es llevado a cabo por Spacy desde un servidor externo para reducir la carga al sistema. Cabe destacar que el análisis llevado a cabo puede ser configurado para que sea superficial, donde solo determina el tipo de palabras a nivel sintáctico, o profundo de forma que genera un árbol de dependencias de la entrada además de llevar a cabo las funciones del análisis superficial. Spacy también interviene en otras funcionalidades de este módulo, como en la eliminación de palabras innecesarias.

```
Texto a analizar: What are the causes of heart attack?
Palabras relevantes extraidas
['cause', 'heart', 'attack']
```

Figura 4.4: Ejemplo de funcionamiento del sistema en la extracción de palabras relevantes.

Cabe destacar que en este módulo también se usa Autocorrect sobre la entrada del usuario para rectificar posibles fallos ortográficos (Figura 4.5) por su parte dado el gran peso que tienen para el trabajo futuro, aunque es importante matizar que solo corrige los términos de forma individual sin detectar errores en relación a otros términos de la entrada.

```
----------------------------
Original phrase: John lO-vs Mary
Corrected phrase: John love Mary

----------------------------
Original phrase: Whta is a heArt atack?
Corrected phrase: What is a heart attack

----------------------------
Original phrase: What is hTe intrenet of zhings?
Corrected phrase: What is the internet of things
```

Figura 4.5: Ejemplo de funcionamiento de Autocorrect.

El resultado obtenido de este componente es un grafo semántico el cual se utiliza en gran parte de los demás componentes del sistema con el fin de que cada uno realice su función correctamente.

Por otro lado, para generar los grafos semánticos a partir del árbol de análisis sintáctico utilizamos Grafeno. Este grafo será utilizado por gran parte del sistema.

No existen, actualmente, muchas tecnologías eficientes con la capacidad de resolver este problema. Incluso Spacy está trabajando en la implementación de esta funcionalidad para su tecnología.

4.3.3. Módulo de obtención de datos para la base de conocimiento

Es el encargado de añadir información a la base de conocimiento para que se puedan responder correctamente a las consultas realizadas por los usuarios.

Como se ha explicado anteriormente en el apartado "Resolución de problemas en múltiples dominios", este componente está implementado para funcionar sobre distintas fuentes de información sin que el resto del sistema se vea afectado por ello. De este modo podemos adaptarnos a distintos campos con solo reemplazar las fuentes de las que obtiene información. Para cada fuente disponemos de una funcionalidad concreta a la que llamamos para obtener los documentos pertinentes en relación a la cuestión del usuario. En el caso de Simple Wikipedia y MedLine realizamos una consulta a través de sus respectivas *APIs* usando los resultados obtenidos del análisis de la cuestión proporcionada por el usuario. A continuación obtenemos una colección de documentos los cuales clasificamos para seleccionar el mejor de ellos como información a introducir en la base de conocimiento relevante.

En el caso de la base de datos en MongoDB partimos de un volcado de un proyecto anterior, el cual constaba de una base de datos SQL con numerosas tablas. Llegados a este punto tuvimos serios problemas para acceder a la información dado que no había ningún tipo de documentación al respecto más allá de los datos de la propia conexión.

Nos supuso un gran esfuerzo identificar como estaba estructurada la información, y tras hacerlo procedimos a volcar la información relevante a una base de datos en MongoDB para trabajar con mayor comodidad y eficiencia.

En este caso disponemos de una colección de información donde, al igual que antes, obtenemos el documento que mejor encaja con la cuestión del usuario. Esta base de datos dispone de un índice creado en base al contenido de todos los documentos de forma que al realizar una *query* tratamos de encontrar la mayor cantidad de términos de búsqueda en un mismo documento. Una vez hecho esto obtenemos el documento con mayor *score* el cual añadimos más adelante a la base de conocimiento.

4.3.4. Base de conocimiento relevante

Esta parte del sistema modela la base de conocimiento que usará el sistema para obtener la información necesaria para generar las respuestas a las consultas.

Debido a la naturaleza del problema a tratar, no todo tipo de base de datos era adecuado. Nuestro sistema trabaja sobre información contenida en textos en lenguaje natural, de modo que era necesario almacenar tanto las expresiones completas que forman el texto como los términos de las oraciones y las relaciones que existen entre ellos.

Por este motivo optamos por modelar nuestro conocimiento mediante grafos, los cuales nos permiten representar de forma eficiente y adecuada el texto que contiene la información que utilizará nuestro sistema.

La tecnología que se adecúa más a nuestras necesidades es Neo4J, una base de datos basada en grafos. Esta nos permite almacenar todas las dependencias y relaciones entre los términos que conforman el texto y realizar consultas sobre ellos.

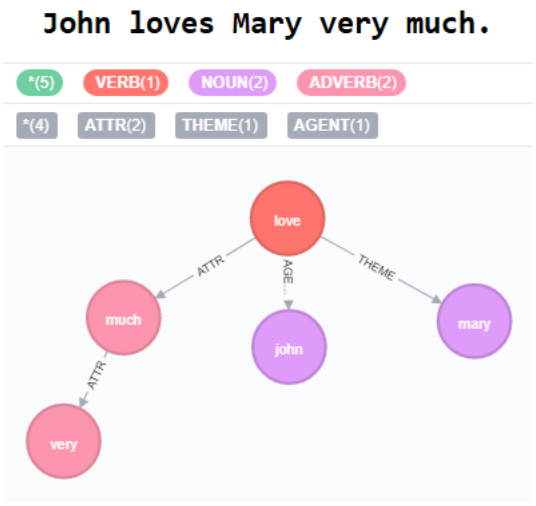


Figura 4.6: Ejemplo de almacenamiento en la base de conocimiento relevante.

4.3.5. Módulo de interacción con la base de conocimiento

Es el encargado de gestionar la comunicación con la base de conocimiento relevante y de realizar todas las peticiones necesarias a la misma, ya sea añadir información obtenida de las distintas fuentes de datos, eliminar información o realizar una consulta que responda a la cuestión realizada por el usuario.

Actúa como un intermediario entre la base de conocimiento relevante y el resto del sistema, luego se realizan peticiones a este módulo para tratar con la base de conocimiento relevante. El resto del sistema, tal y como se ha visto en apartados anteriores, trabaja con cadenas de caracteres o con el resultado obtenido del procesamiento realizado por el módulo de análisis, por lo tanto este módulo tiene la capacidad de, utilizando el resultado de dicho análisis, generar consultas para la base de conocimiento. Estas consultas cumplen las reglas de acceso que impone la tecnología utilizada en la base de conocimiento relevante y pretenden ser equivalentes al resultado del análisis descrito anteriormente.

Tal y como se comentaba en el apartado anterior, la tecnología utilizada para representar la base de conocimiento relevante es Neo4J. Para realizar consultas a una base de datos Neo4J se utiliza el lenguaje Cypher, de manera que este módulo es capaz de generar automáticamente peticiones en este lenguaje para responder a las necesidades de los demás módulos usando los grafos semánticos proporcionados por dichos módulos. Para generar dicha *query* usamos Grafeno y en función del tipo de petición a realizar se genera un tipo de *query* u otro (Figuras 4.7 y 4.8).

```
Query de inserción generada con texto "John loves Mary very much"
CREATE (n:VERB {concept: 'love', sempos: 'v', tense: 'None', polarity: '+', ground_id: '1', _temp_id: '0'});
CREATE (n:NOUN {concept: 'john', sempos: 'n', proper: 'False', num: 'p', ground_id: '1', _temp_id: '1'});
CREATE (n:NOUN {concept: 'mary', sempos: 'n', proper: 'False', num: 'p', ground_id: '1', _temp_id: '2'});
CREATE (n:ADVERB {concept: 'much', sempos: 'r', ground_id: '1', _temp_id: '3'});
CREATE (n:ADVERB {concept: 'very', sempos: 'r', ground_id: '1', _temp_id: '4'});
MATCH (n {_temp_id: '0'}), (m {_temp_id: '1'}) CREATE (n)-[r:AGENT {weight: '1.0'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '2'}) CREATE (n)-[r:THEME {weight: '1.0'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '3'}) CREATE (n)-[r:ATTR]->(m);
MATCH (n {_temp_id: '3'}), (m {_temp_id: '4'}) CREATE (n)-[r:ATTR]->(m);
MATCH (n) REMOVE n._temp_id;
```

Figura 4.7: Ejemplo de *query* de inserción generada por Grafeno.

```
Query de consulta generada texto "What are the causes of heart attack?"
MATCH
(x0:VERB {concept: 'be', sempos: 'v', polarity: '+', ground_id: '1'}),
(x1:NOUN {concept: 'cause', sempos: 'n', ground_id: '1'}),
(x2:NOUN {concept: 'attack', sempos: 'n', ground_id: '1'}),
(x3:NOUN {concept: 'heart', sempos: 'n', ground_id: '1'}),
(x0)-[:COP {weight: '1.0'}]->(what),
(x0)-[:COP {weight: '1.0'}]->(x1),
(x1)-[:COMP {class: 'of', pval: 'of'}]->(x2),
(x2)-[:ATTR]->(x3)
OPTIONAL MATCH path = (what)-[*..4]->()
RETURN DISTINCT what, path
```

Figura 4.8: Ejemplo de *query* de consulta generada por Grafeno.

Además, se han realizado modificaciones adicionales sobre Grafeno que nos permiten separar el contenido de la propia base de conocimiento relevante en distintas conversaciones. La información correspondiente a una conversación solo se utilizará en esa conversación, aunque todo el conocimiento se almacene en la misma base de datos (Figura 4.9).

```
ID de la primera conversación: 4
ID de la segunda conversación: 5
----------------------------------------
Añadimos información en la conversación con id 4
La información añadida es "John loves Mary"
----------------------------------------
Preguntamos desde la conversación con id 4
La pregunta realizada es: John loves Mary
La respuesta obtenida es:
John loves mary.
----------------------------------------
Preguntamos desde la conversación con id 5
La pregunta realizada es: John loves Mary
La respuesta obtenida es:
Sorry, I don't know the answer.
```

Figura 4.9: Ejemplo de comportamiento del sistema frente a distintas conversaciones.

Por último, es importante matizar que el conocimiento adquirido en las distintas conversaciones se elimina cuando la sesión asociada a dicha conversación finaliza.

Capítulo 5

Evaluación del sistema sobre varios dominios

Resumen: en este capítulo hablaremos del comportamiento del sistema implementado frente a diversos dominios de distinta complejidad y de los resultados obtenidos en cada uno de ellos.

5.1. Introducción

Una vez finalizado el proyecto decidimos probarlo usando diferentes fuentes de información o dominios, cada una de ellas con diferente complejidad con el fin de evaluar el funcionamiento del sistema.

Las fuentes de datos utilizadas son:

- **Repositorio propio:** selección de preguntas y respuestas de distintos ámbitos seleccionados dentro de una multitud de sitios web. Dicho repositorio consta de texto de complejidad lingüística baja y contiene las respuestas a las consultas de forma explícita en los textos que lo conforman.
- **Simple Wikipedia:** selección de artículos de Simple Wikipedia que, tal y como se ha explicado en apartados anteriores, contiene artículos con una estructura más simple que la Wikipedia estándar luego son más fáciles de tratar.
- **BioASQ Challenge:** selección de preguntas y respuestas de entrenamiento propuestas por BioASQ Challenge. Estas preguntas presentan una estructuras sintácticas dispares y, en muchos casos, la respuesta no se encuentra de forma explícita en los textos que proporcionan

para su resolución, lo que dificulta considerablemente la búsqueda de respuestas.

5.2. Repositorio propio

Las primeras pruebas del sistema se realizaron sobre un repositorio creado por nosotros. Dicho repositorio contenía un conjunto de 5 preguntas, cada una asociada a uno o varios textos (*snippets*) que podían contener la respuesta (al menos uno de ellos la contenía). Estos textos fueron extraídos manualmente de distintos sitios web y contenían información de distintos ámbitos.

Este repositorio se creó con la finalidad de comprobar el correcto funcionamiento del sistema, que frente a preguntas debería tener la capacidad de responder correctamente en la mayoría de los casos. Estas son preguntas con una estructura sintáctica simple, aquellas que muestran de forma explícita la respuesta en los propios textos, etc.

Por ejemplo, en la figura 5.1 se muestra una de estas preguntas. Como se puede observar, la respuesta asociada se encuentra explícitamente en el texto, es decir, no aparece a través de un sinónimo, mediante un pronombre, etc.

```
1- "YouTube is an American video-sharing website
headquartered in San Bruno, California."
-------------------------------------
Question: What is YouTube?
Question type: open
Ideal answer:YouTube is an American video-sharing website.
Answer: American website is youtube.
-------------------------------------
```

Figura 5.1: Ejemplo de pregunta respondida correctamente. En la parte superior se muestran los textos (*snippets*) de los cuales se obtiene la respuesta y abajo cómo responde el sistema.

Tras probar el sistema con las preguntas y textos que componían el repositorio propio, los resultados pueden considerarse satisfactorios, respondiendo correctamente 4 de las 5 preguntas propuestas, tal y como se muestra en la figura 5.2.

Podemos comprobar como, a pesar de que la construcción de la sentencia en lenguaje natural no es la misma que haría un ser humano, el contenido devuelto responde satisfactoriamente la consulta introducida por el usuario.

Own test cases

```
do_tests('Datasets/Own-Sample.json', 'Datasets/Own-Answer.json')
```

```
-------------------------------------
Question: What is the largest planet?
Question type: open
Ideal answer:Jupiter is the largest planet in the solar system
Answer: Time giant planet with thousandth that sun mass is large planet.
-------------------------------------
Question: What is YouTube?
Question type: open
Ideal answer:YouTube is an American video-sharing website.
Answer: American website is youtube.
-------------------------------------
Question: Who founded Google?
Question type: open
Ideal answer:Google was founded by Larry Page and Sergey Brin.
Answer: Sorry, I don't know the answer.
-------------------------------------
Question: What is the meaning of life?
Question type: open
Ideal answer:There is no answer to this question
Answer: Nothing is meaning of life.
-------------------------------------
Question: Who is the richest person in the world?
Question type: open
Ideal answer:The richest person in the world is Jeff Bezos with 112 billion
Answer: Jeff bozos is rich person in world.
```

Figura 5.2: Ejemplo de ejecución del sistema respondiendo a preguntas del Repositorio propio.

La única pregunta que no responde correctamente es la que se muestra en la figura 5.3. Evaluando el contenido del texto donde se encuentra la respuesta, podemos observar que la causa del error es que la respuesta aparece en el texto de forma pasiva, la cual es una construcción verbal que Grafeno, actualmente, no procesa correctamente.

Para responder correctamente a preguntas como esta, sería necesaria añadir funcionalidades a Grafeno para que procese correctamente sentencias en pasiva.

```
1- "Google was founded by Larry Page and Sergey Brin in 1998."

 ------------------------------------
 Question: Who founded Google?
 Question type: open
 Ideal answer:Google was founded by Larry Page and Sergey Brin.
 Answer: Sorry, I don't know the answer.
 ------------------------------------
```

Figura 5.3: Ejemplo de pregunta respondida incorrectamente. En la parte superior se muestran los textos (*snippets*) de los cuales se obtiene la respuesta y abajo cómo responde el sistema.

5.3. Simple Wikipedia

Simple Wikipedia es una versión de Wikipedia la cual se creó con la idea de facilitar la lectura a niños, gente que está aprendiendo el idioma o personas con dificultades para leer o aprender. Para ello se han evitado términos o artículos complejos de la Wikipedia original y se han empleado las palabras más comunes del inglés y frases más sencillas a la hora de escribir los artículos.

Aunque en Simple Wikipedia se simplifiquen las estructuras sintácticas y semánticas de la Wikipedia estándar, el uso de esta fuente de conocimiento es de especial interés debido a que los artículos de Simple Wikipedia presentan un mayor porcentaje de aparición de formas o construcciones verbales que dificultan la tarea del sistema (como, por ejemplo, la forma verbal pasiva) y gran parte de las respuestas de las preguntas no se encuentran de forma explícita en en texto (5).

Para evaluar el comportamiento del sistema hemos seleccionado un total de 7 preguntas cuyas respuestas se encontrasen en artículos de Simple Wikipedia. Cada una de estas preguntas tiene asociada varios textos (*snippets*) extraídos de Simple Wikipedia donde se encuentran las respuestas a dichas preguntas.

A diferencia del apartado anterior, la evaluación utilizando el repositorio propio, obtenemos los textos donde se encontrarían las respuestas a las preguntas correspondientes utilizando la API de Simple Wikipedia. Mediante consultas a dicha API, Simple Wikipedia nos devuelve un conjunto de textos que podrían contener la respuesta a la pregunta concreta. Estos textos son los *snippets* a los que hemos hecho referencia anteriormente y se almacenarán en la base de conocimiento relevante para responder a la pregunta.

Los resultados obtenidos distan notablemente en función de la pregunta realizada:

Por un lado, las preguntas cuyos textos proporcionados por la API de Simple Wikipedia contienen la respuesta de forma explícita se responden correctamente, tal y como se ve en la figura 5.4.

```
1- "This Is What You Came For" is a song written and
 produced by Scottish DJ Calvin Harris. It features
 Rihanna. The song was released on May 1, 2016, via"

2- "the YouTube channel TwistedFalcon, scored the film
 an A and stated "The screenplay of this movie is so
 phenomenal." TwistedFalcon (2017-12-03), What If"

3- "built after three 1986 tube stock prototype trains
 were tested to find out what the public wanted on the
 new trains. This short article about transport"

4-"Sciences. Retrieved February 6, 2017.  "ichnospecies".
 Oxford Reference. Oxford University Press. Retrieved
 February 6, 2017.  What is Ichnology? YouTube"
 -------------------------------------
 Question: What is YouTube?
 Question type: open
 Ideal answer:  YouTube is an American video-sharing website
 headquartered in San Bruno, California
 Answer:  Video sharing free website is youtube.
 Video sharing free website is youtube.
 Free video sharing website is youtube.
 Free video sharing website is youtube.
 -------------------------------------
```

Figura 5.4: Ejemplo de pregunta respondida correctamente. En la parte superior se muestran los textos (*snippets*) de los cuales se obtiene la respuesta y abajo cómo responde el sistema.

En la figura anterior podemos apreciar como las múltiples sentencias devueltas, a pesar de ser distintas, todas tienen una construcción similar y un contenido que responde satisfactoriamente la pregunta.

Por otro lado, existen numerosas (un total de 7) preguntas para las cuales los textos proporcionados por la API de Simple Wikipedia no contienen la respuesta. Como los textos proporcionados por Simple Wikipedia no contienen la respuesta a la pregunta, el sistema no puede generar ninguna respuesta.

Tal y como se puede observar en la Figura 5.5, el sistema no responde correctamente a la pregunta pero los *snippets* proporcionados por la API de Simple Wikipedia no contienen la respuesta a la propia pregunta propuesta por ellos, de modo que nuestro sistema no puede inferir la respuesta.

```
1- "The heart is an organ found in every vertebrate.
It is a very strong muscle that is about the size
of a fist. It pumps blood throughout the body.
It has"

2- "call 9-1-1 because they think they are having
a heart attack or a nervous breakdown. Though panic
attacks make people feel terrible, they are not
dangerous"

3-"Heart block is a type of heart disease. The human
heart uses electrical signals to make the heart beat.
Electricity travels down pathways in the heart"

4- "the brain. The most common cause for a stopping
heart is a heart attack. The person helping must:
Make sure there is no danger Get to the patient. Talk"

-------------------------------------
Question: What is a heart attack?
Question type: open
Ideal answer:  A sudden occurrence of coronary thrombosis,
typically resulting in the death of part of a heart muscle
and sometimes fatal
Answer: Sorry, I don't know the answer.
-------------------------------------
```

Figura 5.5: Ejemplo de pregunta respondida incorrectamente. En la parte superior se muestran los textos (*snippets*) de los cuales se obtiene la respuesta y abajo cómo responde el sistema.

En resumen, el porcentaje de acierto, con respecto al apartado anterior, ha disminuido considerablemente. En este caso, no se debe a la falta de capacidad del sistema implementado, sino que la propia API de Simple Wikipedia, en la mayoría de los casos, no ha proporcionado textos que contengan las respuestas a las preguntas.

Para terminar con este apartado, veamos más ejemplos del funcionamiento del sistema implementado frente a preguntas propuestas por Simple Wikipedia (Figura 5.6).

```
Question: What is YouTube?
Question type: open
Ideal answer:  YouTube is an American video-sharing website headquartered in San Bruno, California
Answer:  Video sharing free website is youtube.
Video sharing free website is youtube.

Question: What is the meaning of life?
Question type: open
Ideal answer:  The meaning of life is a philosophical question about the purpose and meaning of lif
e, or of existence more generally
Answer:  Sketch comedy musical british movie is meaning of life.
British musical sketch comedy movie is meaning of life.

Question: Who founded Google?
Question type: open
Ideal answer:  Lawrence Edward Page is an American computer scientist and Internet entrepreneur who
co-founded Google with Sergey Brin
Answer: Sorry, I don't know the answer.

Question: What is pizza?
Question type: open
Ideal answer:  Pizza is a traditional Italian dish consisting of a yeasted flatbread typically toppe
d with tomato
Answer:  Type of food is pizza.
Neapolitan chef is pizza.
```

Figura 5.6: Ejemplo de ejecución del sistema respondiendo a preguntas de Simple wikipedia.

5.4. BioASQ

Tal y como se comentó en la sección de trabajo previo, BioASQ Challenge es un desafío organizado por miembros de distintas universidades que propone diversos retos relacionados con áreas del procesamiento de lenguaje natural y, en general, con la inteligencia artificial en el ámbito biomédico.

Entre estos objetivos hay uno especialmente interesante para nuestras actuales metas. Este es la creación de un sistema de respuestas de cuestiones biomédicas en lenguaje natural. Para ello a todos los participantes se les proporcionan diferentes conjuntos de testeo, en inglés, junto con otra información relevante. Los participantes deben implementar sistemas capaces de responder al mayor número posible de preguntas incluidas en dichos conjuntos.

Para evaluar el funcionamiento del sistema, BioASQ Challenge proporciona un conjunto de preguntas, de las cuales hemos extraído 14, junto a un conjunto de textos (*snippets*) en los que se encuentran las respuestas a dichas preguntas. Es importante matizar que, a diferencia de los casos de prueba utilizados en el repositorio propio donde las preguntas las extraíamos manualmente de los textos seleccionados, BioASQ proporciona tanto las preguntas como los textos para probar las capacidades de los sistemas.

La complejidad lingüística de las preguntas propuestas por BioASQ Challenge es mucho más elevada que la que aparecía en el repositorio propio y en Simple Wikipedia. Las respuestas a las preguntas, en la mayoría de los casos, consisten en la combinación de múltiples aspectos descritos en distintos puntos del texto en el cual se encuentra la respuesta (tal y como se muestra en la figura 5.7) y, además, las respuestas no se encuentran de forma explícita.

```
1- "It therefore appears that the LFS phenotype has been
conferred by an aberrant gene, showing a dominant pattern
of inheritance, which may be acting to compromise normal
p53 function rather than by a mutation in p53 itself."

2- "In addition, there seem to be predispositions to a
wider range of different, but well-defined neoplasms:
e.g., adenocarcinomatosis of the colon and the endometrium,
or the Li-Fraumeni/SBLA syndrome. The latter shows a spectrum
 of sarcoma, brain tumours, breast cancer, leukaemias, lung
 and adenocortical cancer. The genes leading to these types
 of dominantly inherited predispositions appear to be the
 tentatively so-called tumour suppressor genes, for which
 the Rb gene serves as a model"

3- "he Li-Fraumeni syndrome is a rare autosomal-dominant
disease whose hallmark is a predisposition to a wide range
of cancers among members of a family."

4- "Li-Fraumeni Syndrome (LFS) is characterized by
early-onset carcinogenesis involving multiple tumor
types and shows autosomal dominant inheritance."
 ------------------------------------
 Question: What is the inheritance pattern of Li-Fraumeni syndrome?
 Question type: open
 Ideal answer:Li-Fraumeni syndrome shows autosomal dominant inheritance.
 Answer: Sorry, I don't know the answer.
 ------------------------------------
```

Figura 5.7: Ejemplo de pregunta respondida incorrectamente. En la parte superior se muestra parte de los textos (*snippets*) de los cuales se obtiene la respuesta y abajo cómo responde el sistema..

Nuestra manera de testear estos documentos de BioASQ Challenge consistía en generar una base de conocimiento a partir de los *snippets* proporcionados para posteriormente realizar la consulta asociada. Después de esto sólo queda comparar la respuesta obtenida con la respuesta ideal proporcionada por expertos.

Los resultados obtenidos han sido poco satisfactorios. El sistema implementado no tiene la capacidad de obtener respuestas a preguntas con una complejidad lingüística tan elevada.

Como hemos explicado anteriormente, nuestro sistema representa tanto la consulta introducida por el usuario como la información contenida en la base de conocimiento relevante mediante grafos semánticos para inferir cuál es el término que se desconoce en la consulta y luego buscarlo en la información de la base de conocimiento relevante. Para responder a las consultas propuestas por BioASQ Challenge, debido a su complejidad lingüística, sería necesario mejorar la generación de grafos semánticos a partir de sentencias en lenguaje natural (añadiendo funcionalidades a Grafeno e incluso a Spacy)

para que estos grafos incluyesen toda la información contenida en los textos de BioASQ.

También sería necesario incluir un mecanismo de inferencia del contexto al sistema, es decir, un mecanismo que determinase a quién o a qué hace referencia cada pronombre utilizado por el sistema y asociar a cada término almacenado en la base de conocimiento relevante un conjunto de sinónimos, ya que las preguntas propuestas por BioASQ tienden a mostrar las respuestas de forma implícita (mediante el uso de sinónimos, pronombres, etc).

Aún así, debido a la distinción explícita de preguntas abiertas y cerradas que presenta el sistema implementado, el sistema es capaz de responder a varias de las preguntas propuestas por BioASQ Challenge.

Nuestro sistema se rige por la hipótesis de mundo cerrado para las preguntas cerradas, es decir, en el caso de que no encuentre la respuesta a la consulta introducida, infiere que la respuesta es "No", mientras que, para las preguntas abiertas, se rige por la hipótesis de mundo abierto.

Entonces, el sistema responde correctamente la totalidad de las preguntas propuestas por BioASQ que sean cerradas y la respuesta sea "No" (como se ve en la figura 5.8) .

```
-------------------------------------
Question: Is the ACE inhibitor indicated for lung cancer treatment?
Question type: closed
Ideal answer:No, the angiotensin converting enzyme (ACE) inhibitors
are used widely as antihypertensive agents. On the contrary, it has
been suggested that they decrease the risk of some cancers, although
available data are conflicting. One study proposes that captopril co
uld be a promising option for the treatment of lung cancer. Furtherm
ore, angiotensin-converting enzyme (ACE) inhibitors have been shown
to mitigate radiation-induced lung injury in preclinical models
Answer: No
-------------------------------------
```

Figura 5.8: Ejemplo de pregunta propuesta por BioASQ que el sistema responde correctamente.

Por último, en la siguiente figura (figura 5.9) se muestra el resultado de la resolución de distintas preguntas proporcionadas por BioASQ Challenge utilizando nuestro sistema.

```
------------------------------------
Question: Is the ACE inhibitor indicated for lung cancer treatment?
Question type: closed
Ideal answer:No, the angiotensin converting enzyme (ACE) inhibitors
are used widely as antihypertensive agents. On the contrary, it has
been suggested that they decrease the risk of some cancers, althoug
h available data are conflicting. One study proposes that captopril
could be a promising option for the treatment of lung cancer. Furth
ermore, angiotensin-converting enzyme (ACE) inhibitors have been sh
own to mitigate radiation-induced lung injury in preclinical models
Answer: No
------------------------------------
Question: What is the main role of Ctf4 in dna replication?
Question type: open
Ideal answer:Ctf4 coordinates the progression of helicase and DNA po
lymerase alpha.  Mcm10 and And-1/CTF4 recruit DNA polymerase alpha t
o chromatin for initiation of DNA replication. And-1/Ctf4 is therefo
re a new replication initiation factor that brings together the MCM2
-7 helicase and the DNA pol alpha-primase complex, analogous to the
linker between helicase and primase or helicase and polymerase that
is seen in the bacterial replication machinery.
Answer: Sorry, I don't know the answer.
------------------------------------
Question: Can tetracyline affect tooth formation?
Question type: closed
Ideal answer:Tetracycline is incorporated in the teeth during their
formation and leads to their permanent staining. A definite relatio
nship between total dosage and staining and duration of administrat
ion and staining was established; the condition occurred with great
er frequency (in more than one-third of the children) when the tota
l dosage exceeded 3 g. or the duration of treatment was longer than
10 days.
Answer: No
------------------------------------
```

Figura 5.9: Ejemplo de ejecución de las preguntas proporcionadas por BioASQ.

Capítulo 6

Conclusiones y trabajo futuro

Resumen: en este capítulo exponemos las conclusiones extraídas durante el desarrollo del proyecto, así como posibles mejoras que podrían implementarse para ampliar la funcionalidad del sistema o mejorar su eficiencia.

6.1. Conclusión

Una vez hemos concluido el desarrollo de este proyecto académico llevado a cabo durante todo el curso, es el momento de realizar un balance de lo aprendido y de los objetivos cumplidos.

La implementación de un sistema de búsqueda de respuestas como el propuesto en este proyecto ha permitido a los distintos miembros del equipo ampliar considerablemente nuestros conocimientos sobre varias ramas de la informática que desconocíamos o habíamos tratado de forma superficial.

El procesamiento o la generación de sentencias en lenguaje natural es un aspecto de la informática poco tratado en el grado y, en mayor o menor medida, hemos adquirido un conocimiento base sobre el funcionamiento general de estos sistemas y las técnicas actuales que se aplican para implementarlos.

También hemos profundizado más en tipos de bases de datos que no se suelen utilizar a lo largo del grado, las bases de datos basadas en grafos. Este proyecto nos ha permitido indagar más en ellas, en los problemas en los que se utilizan este tipo de bases de datos y en usos concretos de las mismas.

Un aspecto que habíamos tratado de forma considerable en el grado pero que este proyecto nos ha permitido comprenderlo mejor es todo lo referente a la recuperación de información (IR), así como las diferencias y dificultades que pueden aparecer cuando se trabaja en este dominio.

Por último, es importante remarcar que la metodología de trabajo utilizada en este proyecto nos ha concienciado de la verdadera necesidad de utilizar sistemas de gestión de proyectos y de seguir una serie de protocolos para obtener una correcta coordinación del proyecto (como por ejemplo, no hacer *commits* a master).

En cuanto a los objetivos fijados en el planteamiento inicial del proyecto, creemos que los hemos alcanzado, aunque debido a la complejidad del problema y a la falta de experiencia inicial han surgido más dificultades de las esperadas. Estas han ralentizado el desarrollo pero, al final, el sistema resultante cumple con las características esperadas. A pesar de ello todas estas dificultades han hecho que adquiramos un conocimiento y experiencia mayor de técnicas que nos eran totalmente desconocidas.

Por otro lado, nuestro sistema responde correctamente en la mayoría de los casos a preguntas cuya respuesta se encuentra explícitamente en los textos que componen la base de conocimiento relevante. Este es un comportamiento satisfactorio, teniendo el cuenta la complejidad de este proceso y, sobre todo, de la implementación de este. Las consultas cuya respuesta no se encuentra explícitamente en la base de conocimiento relevante requieren un procesamiento adicional el cual no se encuentra en nuestro sistema. Aún así, nuestro programa responde adecuadamente a algunas de estas cuestiones, sobre todo si son preguntas cerradas.

En resumen, el balance ha sido positivo, sobre todo por el conocimiento adquirido durante el desarrollo de este proyecto.

6.2. Conclusion

Once concluded the development of this academic project implemented during the whole year it is time to review what we have learnt and the accomplished objectives.

The implementation of a question answering system as the one proposed in this project has allowed the different members of the team to considerably extend their knowledge about several fields of computer science which were unknown to them or where the group had little experience.

The process and generation of natural language sentences is an area of computer science that receives little discussion in the degree and, to a greater or lesser extent, we have adquired a knowledge base about the general functioning of these systems and the actual techniques applied to implement them.

We have also familiarized with other kind of databased not usually used in the degree, the graph databases. This project has allowed us to investigate them, the problems related to this kind of databases and their applications.

Information retrievement is a topic we had considerably treated during the degree. This project has permitted us to get a greater understanding of the topic and the difficulties of working in this field.

Finally, is important to highlight that the work methodology has made us aware of the necessity of using project management systems and following protocols in order to coordinate the work of the group (for example, not to commit to master branch).

In terms of the objectives set in the initial approach of the project, we believe we have accomplished them, although due to the complexity of the problem and the lack of initial experience more difficulties than the expected appeared. These have slowed the development but, at last, the resulting system meets the expected requirements. Despite this, all the difficulties have made us get a greater knowledge and experience about totally unknown techniques.

On the other hand, our system answers correcty to most questions where the answers can explicitly be found in the texts contained in the relevant knowledge base. This is a correct behaviour given the complexity of the process and especially de implementation of the same. Questions which answer does not explictly appear in the relevant knowledge base require and ad-

ditional processing not implemented in our system. Even so, our program answers correctly to some of these queries, especially to closed questions.

In summary, the balance is positive, especially because of the knowledge adquired during the development of this project.

6.3. Trabajo Futuro

Tras terminar el proyecto necesariamente surge la pregunta ¿y ahora qué más se puede hacer?

Lo cierto es que en el contexto de este trabajo hay una infinidad de mejoras y nuevas ideas que se pueden aplicar, no obstante, a continuación vamos a proceder a explicar algunas de ellas.

6.3.1. Migrar de Python a Cython las partes críticas

Durante la realización de este proyecto han aparecido algunos problemas críticos. Uno de estos es la eficiencia, tanto en memoria utilizada como en tiempos de respuesta. Esta posible mejora pretende resolver el segundo caso.

Podemos ver que toda la implementación tanto del chatbot como de Grafeno está realizada en Python. Esto nos ofrece una gran cantidad de ventajas: simplicidad, legibilidad de código y compatibilidad con las librerías de procesamiento de lenguaje natural actualmente existentes.
Pero también tiene una gran desventaja; en comparación con otros lenguajes fuertemente tipados y compilados (en lugar de interpretados), los tiempos de respuestas de Python son sustancialmente peores.

En este sentido, C es el lenguaje con mejores tiempos ejecución, es por esto que surgió Cython, tratando de juntar las mejores partes de Python y C.

Gracias a Cython, mediante un lenguaje muy parecido a Python puedes generar automáticamente código C que es fácilmente importado desde cualquier otro *script* Python.

Esta filosofía de trabajo está actualmente muy extendida en el ámbito de la Inteligencia Artificial y el procesamiento de lenguaje natural. De hecho algunas de las librerías que hemos usado durante este proyecto, tales como Spacy están hechas parcialmente en Cython.

Por ello, nuestra propuesta consiste en convertir a Cython las partes críticas y de cómputo más costoso.

Por último, y para acreditar todo lo mencionado anteriormente, hemos realizado una pequeña prueba que consiste en un bucle con 500 iteraciones en Python y exactamente el mismo bucle hecho en Cython añadiendo tipado a las variables. Además, para mayor exactitud, este test lo hemos realizado 100 veces y hemos calculado la media de los tiempos que ha tardado en ambos casos.

El resultado obtenido es realmente satisfactorio, y Cython pasa a ser aproximadamente un 400 % más rápido que Python, aunque esto por supuesto es algo que depende del problema.

El motivo de esta mejora es que Python comprueba el tipo de cada variable antes de cada uso de la misma, luego realiza un total de 500 preguntas, mientras que en C realiza dicha comprobación una única vez.

6.3.2. Implementar una aproximación sencilla de contexto

Una vez implementada la conservación del conocimiento dentro de una conversación con el fin de evitar la búsqueda de información redundante, también sería interesante dotar al sistema de una noción de contexto. Esta es una de las tareas más complicadas a realizar dentro del campo de procesamiento de lenguaje natural ya que no es algo estrictamente parametrizable. Con esta mejora el usuario podría encadenar consultas de manera sencilla sin tener que estar refiriéndose explícita y continuamente al sujeto.

Usuario: Who are The Beatles?
Usuario: Where are they from?

En este caso el sistema debería establecer una conexión entre “they” y “The Beatles”.

Usuario: John loves Mary?
Usuario: Does he love her?

En este caso deberían crearse las asociaciones “he - John” y “her - Mary”.

En una primera versión podría implementarse mediante reconocimiento del género y número del sujeto y el complemento directo (teniendo en cuenta que este es el sujeto en la forma pasiva). Con esta información se procedería a la comparación y sustitución con los pronombres en caso de su aparición.

Podríamos almacenar cuatro referencias, una por cada combinación de género y número, guardando siempre la última palabra correspondiente. Las referencias se guardarían asociadas a la conversación una vez resuelta la consulta y generada la respuesta.

En caso de existir referencias con las que sustituir los pronombres de la consulta se preprocesaría el texto antes de realizar su análisis.

Para resolver este problema necesitaríamos:

- Almacenar las referencias con las que sustituiríamos los pronombre.
- Consultar el género y número de las palabras.
- Procesar las consultas antes de analizarlas.
- Actualizar las referencias después de cada respuesta.

Todo lo mencionado anteriormente puede conseguirse mediante modificaciones en la implementación de las conversaciones.

Teniendo en cuenta la ambigüedad del lenguaje natural tendremos que resolver las siguientes excepciones:

- Los términos masculino y femenino en plural podrían utilizar una misma referencia ya que ambos compartirían el pronombre “them”.
- Las referencias del masculino y femenino singular se asociarían únicamente con nombres propios, ya que el resto de sustantivos se corresponderían con el pronombre “it”.

Para referencias plurales podrían almacenarse tanto términos como listas de palabras. En caso de guardar una lista se complicaría el preprocesado de la consulta ya que deberíamos computar la conjunción/disyunción. En cada lista deberíamos incluir la relación entre los términos que la componen. Por ejemplo:

Usuario: John likes tomatoes and strawberries?
Usuario: What color are them?

En este caso las referencia serían [‘tomatoes’, ‘strawberries’, ‘and’] como plural y ‘John’ como masculino singular.

6.3.3. Devolución de posibles enlaces de interés

Actualmente la capacidad de comprensión de nuestro sistema es limitada. Es frecuente, sobre todo cuando las consultas se formulan con una estructura compleja, que nuestro sistema no encuentre la respuesta o que directamente no genere una respuesta apropiada.

Por ello sería de especial interés que nuestro sistema proporcionase al usuario un conjunto de enlaces en los cuales podría encontrar la respuesta a su consulta o, sencillamente, ampliar la información proporcionada.

Respecto a esta tarea, aparecen dos preguntas clave: cuándo se ofrecerá esta funcionalidad y cómo se implementará.

Respondiendo a la primera pregunta, por un lado esta funcionalidad podría ofrecerse solo cuando el sistema no obtenga una respuesta. De esta forma, el sistema proporcionaría al usuario un enlace donde podría estar la respuesta que no supo encontrar, a modo de parche de un déficit del sistema.

Por otro lado, el sistema podría proporcionar estos enlaces siempre, encuentre una respuesta o no. Estos enlaces podrían justificarse, aunque el sistema encontrase una respuesta, como información adicional a la respuesta proporcionada.

Ambas opciones son válidas pero aunque no lo parezca, confrontan visiones del sistema muy opuestas.

Al proporcionar siempre los enlaces, estamos devaluando la funcionalidad principal del sistema, que consiste en generar respuestas acordes a las consultas de los usuarios. Estas respuestas, deberían contener todo lo que el usuario busca saber cuando realiza la consulta y, al ofrecer enlaces que contienen más información, estamos infravalorando la respuesta generada.

La otra opción, proporcionar siempre los enlaces, permite que los usuarios encuentren la respuesta que buscan con mayor facilidad y rapidez, lo que también es un factor a tener en cuenta.

En resumen, cuándo ofreceríamos estos enlaces sería un aspecto a discutir entre los componentes del equipo de desarrollo en el futuro y dependería, sobre todo, del ámbito al cual se destinase nuestro sistema y a la capacidad de generación de respuestas del chatbot.

6.3.4. Memoization

Memoization es una técnica de optimización para agilizar programas almacenando los resultados de cómputos costosos en local con el fin de evitar recalcularlos en un futuro, devolviendo el valor almacenado en este caso.

Una de las características a destacar de nuestro sistema es el elevado tiempo de respuesta frente a las cuestiones de los usuarios. En este marco nos encontramos con que las respuestas tardan en procesarse más de 20 segundos lo cual ya de por sí es bastante. Por ello pensamos en introducir una serie de mejoras en nuestro sistema con el objetivo de paliar este déficit.

Para ello modificaremos el módulo de interacción con la base de conocimiento con el fin de almacenar todas las consultas de las que se ha obtenido respuesta. En es caso de las respuestas solo nos quedaremos con el resultado final, omitiendo los documentos relevantes y demás información usada para obtener la respuesta. Respecto de la pregunta tenemos varias formas de hacerlo:

- tomar la pregunta sin ningún tipo de procesamiento: esta es la opción más simple aunque sin duda la menos eficaz ya que obligamos a que el usuario realice dos veces una consulta idéntica, lo cual reduce en gran medidas las posibilidades de ahorrarnos los cómputos costosos.
- tomar los términos relevantes de la pregunta: en este caso nos quedaríamos solo con los términos resultados del análisis de la pregunta de forma que es más probable ahorrarnos los cálculos al disponer de información más concreta.

Capítulo 7

Contribuciones individuales

7.1. Aitor Cayón Ruano

Este proyecto comenzó con una fase de investigación en la cual buscamos información sobre las tecnologías existentes, los sistemas de búsqueda de respuestas y el procesamiento del lenguaje natural.

Para cubrir en mayor profundidad la investigación cada miembro se hizo responsable de distintos campos. En mi caso, fui el encargado de comprobar las capacidades de la primera potencial tecnología a utilizar, PostgreSQL. La prueba de concepto realizada se enfocó en estudiar la eficacia del procesado de lenguaje natural de dicha herramienta. Los resultados fueron limitados y PostgreSQL fue descartada como una opción.

Al no tener experiencia previa con bases de datos basadas en grafos también me encargué de investigar las funcionalidades proporcionadas por Neo4J y Cypher, el lenguaje utilizado en sus consultas. La adecuación a nuestras necesidades y la facilidad de uso de sus sintaxis hicieron que nos decantásemos por esta tecnología.

Una vez completada la fase de investigación procedimos a la implementación de nuestro sistema. Partíamos de Grafeno, una librería desconocida para el equipo, lo que suponía un esfuerzo inicial de estudio y comprensión. Para facilitar el trabajo de mis compañeros fui el encargado de analizar el código, localizando aquellas clases relevantes para nuestro sistema y obteniendo una visión global de dónde se realizaba cada tarea.

La primera *issue* que precisó de una modificación de Grafeno fue la distinción explícita de las preguntas. Esta se comenzó a implementar de manera conjunta de modo que todos los miembros del equipo se familiarizasen con la librería. Esta aproximación nos proporcionó una comprensión más sólida del

funcionamiento de los linearizadores y de los transformadores que componen Grafeno.

Mis principales aportaciones en el proyecto han sido en *issues* relacionadas con la modificación de Grafeno, ya que al haber realizado el análisis inicial del código me sería más sencillo localizar donde implementar las modificaciones.

Fui el encargado de asignar identificadores a las conversaciones. La versión de Grafeno de la que partíamos estaba implementada de tal manera que, ante la inserción, borrado o consulta de información, estas se ejecutaban indistintamente sobre todas las conversaciones. Esto podía acarrear tanto pérdida de información como la replicación innecesaria de conocimiento.

Como solución a este problema cada vez que se crea una conversación se le asigna un identificador único. Cada vez que se inserta o elimina información de la base de conocimiento la tarea se ejecuta sobre la información que coincida con el identificador de la conversación.

Finalmente, y aprovechando el conocimiento adquirido sobre Grafeno mediante el análisis del código y la implementación de las mejoras también, elaboré junto a mi compañero Fernando el *script* de visualización para que el usuario pueda comprobar de manera sencilla el funcionamiento de nuestro sistema.

7.2. Gabriel Sellés Salvà

Para hablar de mi contribución en este proyecto es importante diferenciar entre la fase de investigación realizada al inicio del proyecto y la fase de implementación del sistema de búsqueda de respuestas.

En la primera fase del proyecto, a cada miembro del equipo se nos asignó uno o varios campos de la informática a investigar para así obtener información relevante para la implementación del sistema en un futuro. En mi caso, mis investigaciones se centraron en el procesamiento y en la generación de sentencias en lenguaje natural.

Estos campos eran totalmente desconocidos tanto para mí como para los demás miembros del equipo, lo que supuso una dificultad adicional a la hora de encontrar las primeras tecnologías relevantes o simplemente, encontrar sitios de interés que mostrasen información útil pero adaptada a nuestro nivel.

La primera tecnología asociada al procesamiento de lenguaje natural investigada fue *TextBlob*. Fui el responsable de realizar pruebas de concepto y de capacidad de esta tecnología para evaluar si era útil para nuestro sistema.

Luego encontramos una tecnología que se ajustaba más a las necesidades del sistema a implementar: *Spacy*. Junto a otros miembros del equipo implementamos varias pruebas de concepto de la tecnología.

Por último investigué también las bases de datos Neo4J (y las *querys* en lenguaje Cypher) y PostgreSQL, aunque mi aportación en la investigación en estos dos aspectos fue inferior que en TextBlob, Spacy y, en general, el procesamiento y generación de sentencias en lenguaje natural.

En la fase de implementación del sistema, mis aportaciones abarcan distintos módulos del sistema.

Fui el encargado, junto a otros miembros del equipo, de añadir a la implementación base del sistema aspectos de conservación de la información de la base de conocimiento relevante y de la modificación del sistema para la eliminación de la fase de búsqueda de información en fuentes de conocimiento cuando esta no fuese necesaria.

En cuando al procesamiento de la entrada introducida por el usuario, participé en la modificación de Grafeno para la distinción de forma explícita entre preguntas abiertas y cerradas. Al modificar Grafeno, también era necesario actualizar ciertos aspectos del módulo de Análisis de sentencias en lenguaje natural y del módulo de interacción con la base de conocimiento relevante. Esta fue una de mis tareas realizadas.

Por otro lado, junto a mi compañero Fernando, nos encargamos de añadir mecanismos de corrección ortográfica para las consultas introducidas por el usuario al sistema.

También participé en el preparativo de las pruebas de capacidad del sistema utilizando las preguntas y textos propuestos por BioASQ Challenge.

7.3. Fernando Pérez Gutiérrez

Durante todo este proyecto y sobre todo durante todo el primer cuatrimestre optamos por dividir las tareas a realizar en grupos, por lo que en la mayoría de estas no fue un trabajo completamente individual, no obstante, procederé a explicar los puntos más relevantes en los que trabajé.

Lo primero en lo que me centré a la hora de afrontar este proyecto fue en buscar proyectos parecidos y realizar una primera aproximación al procesamiento del lenguaje natural. En este último punto, me centré en crear una serie de pruebas de concepto utilizando las librerías *Spacy* y *NLTK* para generar árboles de análisis sintácticos y representarlos gráficamente.

Por otro lado realicé pruebas de concepto para realizar las conexiones de Python a *PostgreSQL*, el sistema de gestión de bases de datos que inicialmente íbamos a utilizar.

Llegados a este punto, se nos comunicó sobre la existencia de la librería Grafeno y comencé a estudiarla para ser capaz de entender todo el proceso que realiza en el procesamiento de lenguaje natural. Cabe destacar que esta no fue una tarea trivial, puesto que nos enfrentábamos a muchas técnicas y tecnologías completamente desconocidas para nosotros en este momento y debíamos entenderlo a la perfección porque posteriormente necesitaríamos realizar modificaciones en dicha librería.

A continuación, y centrándome de nuevo en el lenguaje natural, diseñé el servicio API Rest que nos permite por medio de conexiones sencillas obtener todo el procesado sintáctico generado por *Spacy*.

La creación de este servicio nos proporcionaba la capacidad de realizar el procesamiento sintáctico de forma remota. Pero, para poder poder utilizarlo realicé las adaptaciones pertinentes en Grafeno para que este fuera capaz de establecer una conexión con dicho servicio y aprovechar la respuesta devuelta por el mismo con una estructura diferente pero parecida a la generada cuando se usa *Spacy* de manera local.

Por otro lado, realizamos un estudio sobre algunas posibles mejoras que pudiéramos y añadir a nuestro sistema, y tras ello, encontramos la librería *Autocorrect*, que permitía corregir con bastante acierto las faltas ortográficas. Por ello decidimos hacer uso de la misma en nuestro sistema, para conseguir que si el usuario comete pocos errores ortográficos nuestro sistema siga interpretando correctamente el mensaje.

Durante la etapa final, y con el fin de verificar la capacidad de respuesta de nuestro sistema, con la ayuda de mis compañeros cree un *script* de testeo capaz de recibir ficheros de prueba (inicialmente pensado para los de *BioASQ* aunque reutilizado para testear nuestros propios ficheros de prueba).

Más adelante, junto con un compañero, creamos un documento donde se podía visualizar paso a paso el proceso que realiza nuestro sistema. Además este fichero nos permite ver los grafos generados como resultado del análisis sintáctico y semántico de las preguntas o del conocimiento adquirido para dicha conversación.

Finalmente, realizamos un último estudio para tratar de proponer posibles mejoras a nuestro sistema. Tras mucho estudio y análisis de los tiempos de ejecución del mismo, realicé unas pruebas de los tiempos de ejecución de pequeños programas hechos con *Cython* comparándolo con sus equivalentes en *Python*. El objetivo de este estudio era verificar si era interesante convertir las partes críticas de nuestro sistema de *Python* a *Cython* y el resultado obtenido fue que en efecto es una muy buena idea debido a que llegó a aumentar la eficiencia de algunas pruebas de concepto en un 400 %.

Como información adicional considero importante recalcar que durante todo este proceso y en prácticamente nada de lo mencionado anteriormente he trabajado completamente solo, pues siempre he recibido algún tipo de ayuda de mis compañeros. Por otro lado, yo he actuado de la misma manera y he ayudado en mayor o menor medida en bastantes de los problemas en los que se han centrado ellos (como podría ser, por ejemplo, la separación del conocimiento en conversaciones o la conservación de las mismas).

7.4. Jose Javier Cortés Tejada

Nada más empezar el desarrollo del proyecto decidimos distribuir las tareas para así avanzar más rápido. A mi me tocó, junto con Gabriel, investigar tecnologías y herramientas para PLN. Me dediqué principalmente a realizar pruebas de concepto con Texblob y NLTK para ver que nos ofrecían de cara a la implementación.

Más tarde cuando ya sabíamos que usar y que no empezamos a fijar tareas más concretas para cada uno. Yo me quedé con la parte de RI (Recuperación de Información) pues me interesa bastante este ámbito y además tengo bastante control sobre bases de datos.

Poco después Alberto y Antonio nos comentaron sobre Grafeno y nos instaron a usarlo para nuestro proyecto. Es aquí donde empecé a aplicar lo aprendido durante el tiempo que pasé investigando así como mis conocimiento sobre bases de datos, pues optamos por usar MongoDB como principal fuente de información para nuestro sistema. Para ello me basé en recursos de proyectos de otros años de los que conseguí extraer artículos médicos que más adelante fueron volcados a una base de datos no relacional, la cual usaríamos para rellenar la base de conocimiento del sistema.

Tras terminar esta tarea me junté con mis compañeros para trabajar en varias issues que necesitaban modificar Grafeno, pues la primera aproximación se nos hizo bastante cuesta arriba a pesar de estar los cuatro trabajando en ello. Estas consistían en añadir un par de linealizadores a Grafeno con el fin de poder distinguir entre preguntas abertas y cerradas.

Además Gabriel y yo nos encargamos de migrar varios notebooks a ficheros .py pues suponían una carga innecesaria para el entorno de trabajo.

Una vez tuvimos una versión estable del sistema nuestros tutores nos instaron a crear ficheros de prueba para ver que tal funcionaba el asunto. En relación a esto me encargué de elaborar ficheros de prueba con información de Simple Wikipedia. Cabe destacar que, como se comentó a lo largo del capítulo 5, los resultados no fueron especialmente sorprendentes, sin embargo esta parte del desarrollo fue muy interesante pues me permitió comprobar varios déficit del sistema.

Sin duda el más notable fue el tiempo que tardaba en dar una respuesta, es por ello que elaboré un documento de diseño donde trataba de paliar este déficit, pero al estar cerca de los exámenes finales y de la entrega del proyecto dejé esta idea de lado por falta de tiempo.

Apéndice A

Ejemplo de funcionamiento

Resumen: en este apéndice explicaremos el funcionamiento de nuestro sistema por medio de un ejemplo completo guiado.

Comenzamos por crear la conversación mediante la cuál el usuario podrá realizar las consultas al sistema asignándole un identificador único a dicha conversación, de modo que la información almacenada por esta conversación sólo será accesible desde la misma, haciendo así que cada conversación sea completamente independiente de las demás (y siendo posible que lo que en una conversación sea cierto en la otra sea falso).

A continuación, introducimos la pregunta *'John helps Mary?'* y realizamos el análisis sintáctico y posteriormente semántico de la misma. Obteniendo de este modo un grafo completo donde marca *'help'* como verbo, *'John'* como el agente y *'Mary'*, el tema. Como podemos ver en la figura A.1.

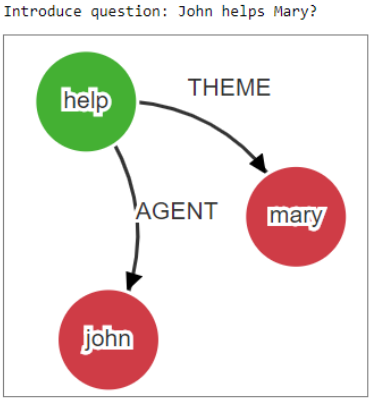


Figura A.1: Análisis obtenido de la pregunta 'John helps Mary?'.

Cabe destacar que el grafo generado es completo debido a que la pregunta es cerrada, es decir, se espera saber si esta afirmación es correcta o no, pero, no hay carencia de información en la misma. Sin embargo, si la pregunta formulada pasara a ser *'Who helps Mary'* el grafo generado sería el mostrado en la figura A.2. Podemos ver que dicho grafo contiene una interrogación debido a la carencia de información en cuanto a quién es el agente que ayuda a 'Mary'.

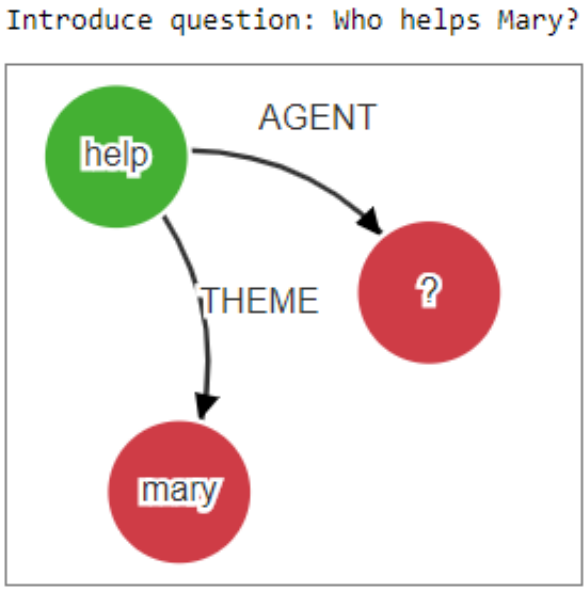
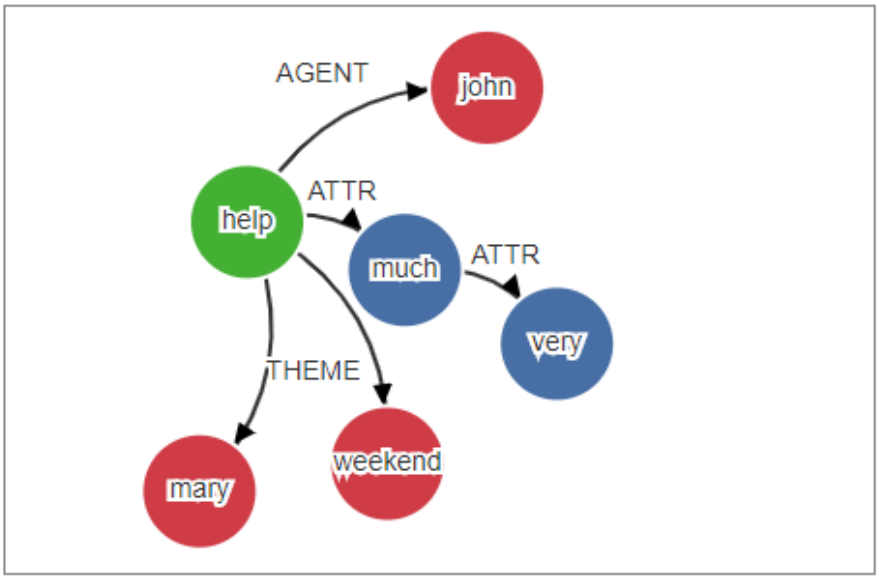


Figura A.2: Análisis obtenido de la pregunta 'Who helps Mary?'.

Una vez aclarado esto continuemos con nuestra pregunta inicial, *'John helps Mary?'*. Para poder responderla necesitamos obtener información de alguna fuente de conocimiento. En este caso y únicamente con fines docentes, optaremos por insertar dicha información de manera manual. De este modo la información que añadiremos será: *'John helps Mary at weekends very much.'* y *'John is very grumpy.'*. Dicha información será analizada tal y como se realizó anteriormente con la pregunta y almacenada en el conocimiento de la conversación actual.

Podemos ver los grafos generados del análisis de ambos fragmentos de información en la figura A.3, donde podemos observar que detecta correctamente los tipos de cada palabra y las relaciones que existen entre las mismas.

```
Introduce snippet: John helps Mary at weekends very much.
```




```
Introduce snippet: John is very grumpy.
```

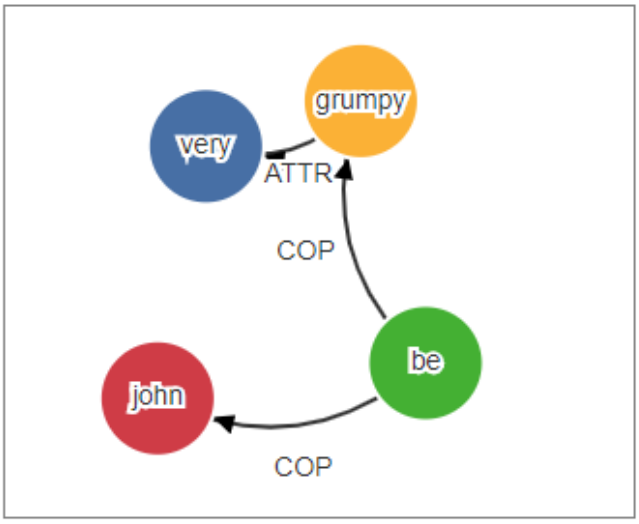


Figura A.3: Información añadida en la conversación actual.

A continuación, debemos añadir dichos grafos a la conversación y almacenarlos en la base de datos Neo4J. Para ello, con la ayuda de Grafeno, se generan las querys mostradas en las figuras A.4 y A.5 haciendo referencia a *'John helps Mary at weekends very much.'* y *'John is very grumpy.'* respectivamente.

```
CREATE (n:VERB {concept: 'help', sempos: 'v', tense: 'None', polarity: '+', ground_id: '0', _temp_id: '0'});
CREATE (n:NOUN {concept: 'john', sempos: 'n', proper: 'False', num: 'p', ground_id: '0', _temp_id: '1'});
CREATE (n:NOUN {concept: 'mary', sempos: 'n', proper: 'False', num: 'p', ground_id: '0', _temp_id: '2'});
CREATE (n:NOUN {concept: 'weekend', sempos: 'n', proper: 'False', num: 'p', ground_id: '0', _temp_id: '3'});
CREATE (n:ADVERB {concept: 'much', sempos: 'r', ground_id: '0', _temp_id: '4'});
CREATE (n:ADVERB {concept: 'very', sempos: 'r', ground_id: '0', _temp_id: '5'});
MATCH (n {_temp_id: '0'}), (m {_temp_id: '1'}) CREATE (n)-[r:AGENT {weight: '1.0'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '2'}) CREATE (n)-[r:THEME {weight: '1.0'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '3'}) CREATE (n)-[r:COMP {class: 'at', pval: 'at'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '4'}) CREATE (n)-[r:ATTR]->(m);
MATCH (n {_temp_id: '4'}), (m {_temp_id: '5'}) CREATE (n)-[r:ATTR]->(m);
```

Figura A.4: Cypher query generada para introducir el análisis del texto: *'John helps Mary at weekends very much.'*.

```
CREATE (n:VERB {concept: 'be', sempos: 'v', tense: 'None', polarity: '+', ground_id: '0', _temp_id: '0'});
CREATE (n:NOUN {concept: 'john', sempos: 'n', proper: 'False', num: 'p', ground_id: '0', _temp_id: '1'});
CREATE (n:ADJECTIVE {concept: 'grumpy', sempos: 'j', ground_id: '0', _temp_id: '2'});
CREATE (n:ADVERB {concept: 'very', sempos: 'r', ground_id: '0', _temp_id: '3'});
MATCH (n {_temp_id: '0'}), (m {_temp_id: '1'}) CREATE (n)-[r:COP {weight: '1.0'}]->(m);
MATCH (n {_temp_id: '0'}), (m {_temp_id: '2'}) CREATE (n)-[r:COP]->(m);
MATCH (n {_temp_id: '2'}), (m {_temp_id: '3'}) CREATE (n)-[r:ATTR]->(m);
```

Figura A.5: Cypher query generada para introducir el análisis del texto: *'John is very grumpy.'*.

Llegados a este punto hemos introducido y analizado la pregunta, así como la información necesaria para obtener la respuesta. Además, debido a que nosotros somos los que hemos introducido la información en el sistema, sabemos que la respuesta ideal a dicha pregunta sería 'Yes'.

Ahora hay que entender cómo nuestro sistema utilizará estos grafos para obtener la respuesta. Para lograrlo, tenemos que realizar un *matching* de ambos grafos. En este caso como la pregunta es cerrada, espera que en los fragmentos de información introducidos aparezca un subgrafo como el de la figura A.1 para responder 'Yes', si no lo encuentra, responderá 'No'.

Por otro lado, si la pregunta fuera abierta, esperaría que aparezca un grafo como el de la figura A.2, pero sustituyendo la 'interrogación' o información ausente, por algún nodo que pueda ocupar dicha posición.

Finalmente, tras todo este proceso, pasamos a la última etapa, en la cuál construimos la respuesta. Para ello, gracias a que se trata de una pregunta cerrada y ha conseguido realizar el *matching*, la respuesta obtenida por nuestro sistema es 'Yes'.

Por otro lado, si fuera una pregunta abierta, necesitaría formular la respuesta en lenguaje natural a partir del grafo generado en el proceso de *matching*.

Apéndice B

Manual de usuario

Resumen: en este apéndice explicaremos aspectos de interés para la instalación y uso del sistema implementado.

Para acceder a nuestro sistema hemos habilitado un repositorio público de GitHub.

Cabe destacar que la implementación de nuestro sistema es compatible con cualquier sistema operativo. A continuación detallaremos todas las librerías y tecnologías necesarias para ejecutarlo correctamente.

- Python[1]: Lenguaje de programación a partir del cuál se ha construido nuestro sistema. La versión necesaria de Python ha de ser 3.4 o superior.
- Grafeno [2]: Librería utilizada para extraer información de los textos, generar grafos y transformar los mismos en sentencias en lenguaje natural. Para su correcto funcionamiento deberán instalarse también todas sus dependencias.
- SpaCy [3]: Genera el análisis sintáctico de los textos. La versión instalada ha de ser 1.9.X.
- Neo4J [4]: Base de datos basada en grafos donde se almacena todo el conocimiento asociado a las conversaciones actuales. Es necesaria la instalación del programa así como la librería Neo4J de Python para interaccionar con la base de datos. Es necesario lanzar la base de datos previamente a nuestro sistema y establecer el mismo nombre de la

[1]Sobre Python: https://www.python.org/
[2]Sobre Grafeno: https://github.com/agarsev/grafeno
[3]Sobre SpaCy: https://spacy.io/
[4]Sobre Neo4J: https://neo4j.com/

base de datos y contraseña que el que se encuentre en el documento *config.yml*.

- Autocorrect [5]: Librería para corregir pequeñas faltas ortográficas que puedan aparecer en la consulta del usuario.

B.1. Recomendaciones para la ejecución del sistema

Como entorno de trabajo para la ejecución de nuestro sistema recomendamos Jupyter Notebook[6], ya que este entorno nos permite ver el código a la vez que lo testeamos y visualizamos las respuestas.

Actualmente existen dos formas de instalar Jupyter Notebook:

- A través de Anaconda: esta distribución nos proporciona varios entornos para trabajar con Python, entre los que se incluye Jupyter Notebook.
- A través de pip: por medio de los comandos mencionados en la Figura B.1.

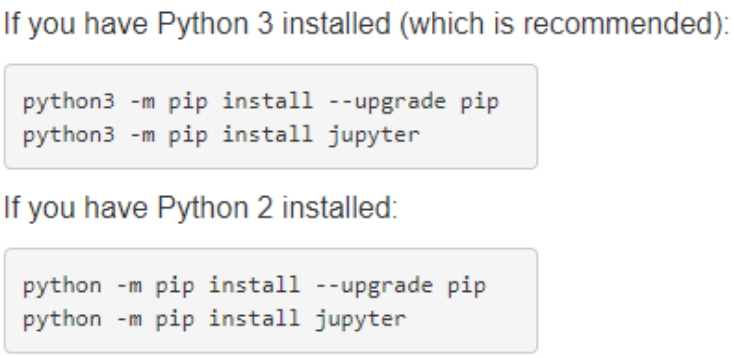


Figura B.1: Instrucciones de instalación de Jupyter Notebook a través de pip.

[5]Sobre Autocorrect: https://github.com/phatpiglet/autocorrect
[6]Sobre Jupyter Notebook: http://jupyter.org/

Nuestra recomendación es que se haga la instalación, tanto de Jupyter Notebook como de todas las dependencias mencionadas anteriormente a través de Anaconda en un entorno [7] creado únicamente para este sistema.

Por último, es recomendable realizar una primera aproximación a nuestro sistema usando el Notebook *ResumenVisualizacion.ipynb*, ya que explica de manera clara y de forma más visual todo el proceso, desde que se hace una consulta hasta que se obtiene la respuesta.

Es importante matizar que este Notebook es un ejemplo de funcionamiento del sistema en el cual se utilizan como fuente de conocimiento los datos introducidos por el propio usuario. Como se comentó a lo largo del documento el sistema tiene la capacidad de trabajar con distintas fuentes de conocimiento, de las cuáles obtiene la información y no es necesario que ésta sea introducida de forma manual. No obstante, como material docente hemos considerado más interesante que el *script* de visualización funcionase así.

Utilizando el Notebook *Pipeline.ipynb* el usuario puede utilizar el sistema seleccionando entre dominios o fuentes de conocimiento de distinta índole: un conjunto de artículos de carácter biomédico de proyectos anteriores y Simple Wikipedia.

Para ejecutar dicho Notebook es necesario ejecutar todas las celdas del mismo y llamar al método *runPipeline* que será el que encargado de solicitar las preguntas y resolverlas.

[7] Anaconda y gestión de entornos: https://conda.io/docs/user-guide/tasks/manage-environments.html

Bibliografía

[1] H. Alshawi, D. Carter, J. van Eijck, R. Moore, and D. Moran. Overview of the core language engine. , University of Cambridge Computer Laboratory and SRI International Cambridge Computer Science Research Centre, 1988.

[2] R. Barskar, G. F. Ahmed, and N. Barskar. An approach for extracting exact answers to question answering (qa) system for english sentences. , CSE Department ,University of Institute Technology, Rajiv Gandhi Prodhyogiki Vishvidhyalaya, Bhopal (M.P.) Pin-462036, India and CSEIT Department Maulana Azad National Institute of Technology, Bhopal (M.P.) Pin-462051, India and CSE Department, University of Institute Technology, Rajiv Gandhi Prodhyogiki Vishvidhyalaya, Bhopal (M.P.) Pin-462036, India, 2012.

[3] S. Bird and E. Loper. Nltk: The natural language toolkit. , University of Melbourne and University of Pennsylvania, 2004.

[4] A. Bouzianea, D. Bouchihaa, N. Doumib, and M. Malkic. Question answering systems: Survey and trends. , Ctr Univ Naama, Inst. Sciences Technologies, Dept. Mathematics Computer Science, Algeria and University Dr. Tahar Moulay of Saida and EEDIS Laboratory, Djillali Liabes University of Sidi Bel Abbes, Algeria, 2015.

[5] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. , Computer Science Stanford University Stanford, CA 94305, USA and Facebook AI Research 770 Broadway New York, NY 10003, USA, 2017.

[6] COMPOSE. Introduction to graph databases. https://www.compose.com/articles/introduction-to-graph-databases/.

[7] Cornell CIS. Intro to natural language processing. https://goo.gl/KjDwBs.

[8] S. K. Dwivedia and V. Singh. Research and reviews in question answering system. , Department of Computer Science, B. B. A. University (A Central University) Lucknow, Uttar Pradesh, India, 2013.

[9] El profesional de la información. Procesamiento de lenguaje natural: bases teóricas y aplicaciones.
https://bit.ly/2v0Q4za.

[10] K. S. Jones. Natural language processing: a historical review. , University of Cambridge, Oxford, UK, 2001.

[11] L. Kodra and E. K. Meçe. Question answering systems: A review on present developments, challenges and trends. , Department of Computer Engineering, Polytechnic University of Tirana, Albania, 2017.

[12] D. Molla and J. L. Vicedo. Question answering in restricted domains: An overview. , Macquarie University, Australia and University of Alicante, Spain, 2007.

[13] J. Prager. Open-domain question–answering. , IBM T.J. Watson Research Center, 1S-D56, P.O. Box 704, Yorktown Heights, NY 10598, USA, 2007.

[14] STANFORD HCI GROUP. Shrdlu.
https://hci.stanford.edu/winograd/shrdlu/.

[15] The Data Incubator. Nltk vs. spacy: Natural language processing in python.
https://blog.thedataincubator.com/2016/04/nltk-vs-spacy-natural-language-processing-in-python/.

[16] G. Tsatsaronis, G. Balikas, P. Malakasiotis, I. Partalas, M. Zschunke, M. R. Alvers, D. Weissenborn, A. Krithara, S. Petridis, D. Polychronopoulos, Y. Almirantis, J. Pavlopoulos, N. Baskiotis, P. Gallinari, T. Artiéres, A.-C. N. Ngomo6, N. Heino, E. Gaussier, L. Barrio-Alvers, M. Schroeder, I. Androutsopoulos, and G. Paliouras. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. , Tsatsaronis et al. BMC Bioinformatics, 2015.